08/01/09

האוניברסיטה הפתוחה

עבודה מסכמת במסגרת לימודי התואר שני
במדעי המחשב

מתודולוגיה לבניית
Use Case Model

מנחה העבודה: ד"ר צבי פירסט ז"ל

מוגש ע"י: דותן בן-אבי , ת.ז.: 032141137

העבודה מוקדשת לזכרו של ד"ר צביקה פירסט אשר ליווה והדריך אותי , היה מוכן לסייע בכל עת ותרם רבות מניסיונו המקצועי רב השנים.

ינואר 2009

תוכן עניינים:

רשימת איורים : 3
רשימת טבלאות : 5
1. תקציר 7
2. מבוא 9
3. שלב I - תיחום גבולות המערכת 21
4. שלב II - זיהוי שחקנים ומטרות 43
5. שלב III - גיבוש Use Case 53
6. שלב IV – הרחבת ה- Use Case 68
7. שלב V – הבטחת איכות המודל 89
8. סיכום : 110
9. נספחים : 113
רשימת מקורות 130

רשימת איורים :

איור מספר 1 – תרשים מתודולוגיית RESCUE 14

איור מספר 2 – תרשים מתודולוגיית ICONIX 15

איור מספר 3 – מרכיבי מתודולוגיית ה Use Case Model המוצעת..... 19

איור מספר 4 – תרשים גאנט עקרוני לביצוע השלבים השונים של המתודולוגיה..... 20

איור מספר 5 – דיאגרמת Context עבור מערכת כספומט בבנק..... 38

איור מספר 6 – תרשים Use Case עבור מערכת כספומט בבנק..... 39

איור מספר 7 – תרשים מסכם לשלב I - תיחום גבולות המערכת..... 41

איור מספר 8 – תרשים מסכם לשלב II - זיהוי שחקנים ומטרות..... 51

איור מספר 9 – תרשים רמות ה Use Case השונות..... 53

איור מספר 10 – שימוש ברשימת שחקנים כאינדקס לחיפוש Use Cases..... 57

איור מספר 11 – תרשים חבילות לדוגמא..... 64

איור מספר 12 – תרשים מסכם לשלב III - יצירת Use Cases שלדיים..... 66

איור מספר 13 – דיאגרמת UC למשיכת מזומנים - דגשי הסתעפויות..... 77

איור מספר 14 – דיאגרמת Sequence המתארת Use Case של משיכת מזומנים מ ATM..... 78

איור מספר 15 – תרשים מסכם לשלב IV - הרחבת ה Use Case..... 87

איור מספר 16 – קשר עקיבות..... 90

איור מספר 17 – מטריצת עקיבות בין Test Cases לבין Use Cases..... 90

איור מספר 18 – היראכיית אפקט השינוי..... 92

איור מספר 19 – ארכיטקטורת 4+1..... 97

איור מספר 20 – Use Case View..... 100

איור מספר 21 – Logical View..... 101

איור מספר 22 –Class Diagram..... 101

איור מספר 23 –Refined Class Diagram..... 102

איור מספר 24 – Statechart Diagram..... 103

איור מספר 25 – Static Structure Diagram..... 103

איור מספר 26 – Implementation View..... 104

איור מספר 27 – Implementation View – Executable Release..... 104

איור מספר 28 – Deployment Diagram..... 105

איור מספר 29 – תרשים מסכם לשלב V – הבטחת איכות המודל..... 108

איור מספר 30 – שחקן..... 125

איור מספר 31 – Use Case 125
איור מספר 32 – יחס בין שחקן ו Use Case 125
איור מספר 33 – יחס הכלה 126
איור מספר 34 – יחס הרחבה 126
איור מספר 35 – יחס הכללה 126
איור מספר 36 – יחס הכללה בין שחקנים 127
איור מספר 37 – גבולות המערכת 127
איור מספר 38 – תרשים Use Case לדוגמה 128

רשימת טבלאות :

טבלה1 השוואה בין המתודולוגיות השונות 16
טבלה2 תיחום גבולות המערכת - שגיאות נפוצות ודרכי התמודדות 25
טבלה3 דוגמא לתוצר רשימת היישויות 28
טבלה4 דוגמא לתוצר הסקופ הפונקציונלי 31
טבלה5 שגיאות נפוצות ביצירה של Use Case Diagrams 36
טבלה6 שגיאות נפוצות בנושאי סקופ וגבולות המערכת 37
טבלה7 תיחום גבולות המערכת - רשימת תוצרים 41
טבלה8 זיהוי שחקנים ומטרות - סיכונים ושגיאות נפוצות 46
טבלה9 דוגמא לתוצר רשימת פרופיל-שחקנים 47
טבלה10 רשימת שחקן-מטרה - סיכונים ושגיאות נפוצות 49
טבלה11 דוגמא לתוצר רשימת שחקן-מטרה 50
טבלה12 דוגמא לתוצר רשימת שחקנים-אחודה 50
טבלה13 רשימת תוצרים – שלב זיהוי שחקנים ומטרות 51
טבלה14 דוגמא לקביעת הרמה בהתאם למטרה 57
טבלה15 טבלה לבקרת איכות עבור סעיפי ה Use Case השלדי 59
טבלה16 גיבוש UC שלדי - סיכונים ושגיאות נפוצות 60
טבלה17 דוגמא לUse Case שלדי 61
טבלה18 חבילות UC - סיכונים ושגיאות נפוצות 64
טבלה19 גיבוש UC - רשימת תוצרים 66
טבלה20 טבלת Pass/Fail עבור Use Case בודד 75
טבלה21 הרחבת UC - סיכונים ושגיאות נפוצות 76
טבלה22 אימות הסתעפויות 82
טבלה23 UC מלאים - סיכונים ושגיאות נפוצות 83
טבלה24 חבילות UC מלאות – סיכונים ושגיאות נפוצות 86
טבלה25 הרחבת ה UC - רשימת תוצרים 87
טבלה26 טבלת בדיקות לתיקוף המודל 91
טבלה27 תיקוף המודל - סיכונים ושגיאות נפוצות 96
טבלה28 שגיאות נפוצות עבור פעילויות הניהול 97
טבלה29 שגיאות נפוצות במהלך ניהול המתודולוגיה 97
טבלה30 מודל UC משולב - סיכונים ושגיאות נפוצות 100

טבלה31 הבטחת איכות המודל - סיכונים ושגיאות נפוצות 107

טבלה32 סיכונים עיקריים בשלבים השונים 108

1. **תקציר**:

Use Case הינו מודל מתחום הנדסת תוכנה אשר מטרתו העיקרית לתאר את הדרישות הפונקציונליות.

הצורך העיקרי בהנדסת תוכנה נובע מכך שמערכות עתירות-תוכנה הפכו כה מורכבות עד כי קשה לנהלן . ככל שאנו משפרים את הכלים ורוכשים ניסיון רב יותר בבניית מערכות תוכנה, כן נפתחים בפנינו תחומי-בעיות רחבים ומורכבים יותר אשר מביאים עמם מורכבות הולכת וגדלה של מערכות התוכנה. הפתרון המקובל לבעיה הוא השימוש בשיטות ובכלים הנדסיים שיעזרו לפתח ולנהל ביעילות מערכות מורכבות כאלה. אוסף השיטות והכלים לפיתוח ולניהול מערכות תוכנה נקרא **הנדסת תוכנה**.

הבנה מלאה של דרישות המשתמשים והלקוחות היא תנאי הכרחי להצלחה של פרויקט פיתוח תוכנה (בקריטריונים של עלות/תועלת). ולכן השלב הראשון ברוב מתודולוגיות הפיתוח הוא **שלב הגדרת ואפיון הדרישות**. המטרה העיקרית של שלב זה היא להגדיר את הדרישות מהמערכת ומתהליך הפיתוח, לפני שמתחילים לתכנן אותה. איכות הדרישות היא גורם קריטי בהצלחה או כשלון של פרויקטי תוכנה. ממצאים מהשטח מראים כי התחקות אחר מקורן של תקלות רבות בפרויקטים יוביל לאחור עד לשלב הדרישות ושם נמצא כי הכיל דרישות לקויות, שגויות ו/או חסרות.

הנדסת הדרישות עוסקת ב "מה" המערכת צריכה לעשות. הנדסת הדרישות כוללת פעילויות רבות אשר מטרתן לחשוף בצורה שיטתית מה על המערכת המפותחת לספק על מנת לענות על צרכי הלקוח ומהם התנאים החיצוניים והאילוצים על המערכת ועל תהליך פיתוחה. הנדסת הדרישות כוללת פעילויות שונות : מיצוי דרישות , ניתוח דרישות , מפרטי דרישות , אימות דרישות, ניהול דרישות ועוד. כחלק מהנדסת הדרישות פותחו מודלים שונים אשר תומכים בפעילויו ת הללו (או חלק מהן). כל מודל עשוי לסווג את הדרישות באופן שונה (דרישות משתמש, דרישות מערכת , דרישות עסקיות , דרישות מוצר, דרישות פונקציונליות, דרישות טכניות , אילוצים , פי"צרים או סתם דרישות). בעבודה זו אנו עוסקים באחד המודלים הללו אשר נקרא **מודל ה Use Case** .

כל Use Case מכיל תרחיש אחד או יותר המתאר כיצד המערכת מתקשרת עם שחקנים בכדי לספק מטרה עסקית מסוימת. טכניקת ה Use Cases נכנסה לשימוש בעיקר ע"י קהילת מפתחי ה Object Oriented אך היא בפירוש איננה מוגבלת למערכות מונחות עצמים בלבד.

בספרות המחקרית והמקצועית קיים מידע רב ומגוון בנושא Use Cases אך מנגד ישנו מחסור במתודולוגיות סדורות ליצירת Use Cases. מרבית החומר המקצועי בנושא עוסק בהנחיות ליצירת Use Case אך פחות עוסק בשיטה כללית וסדורה ליצירת המודל כולו .

עבודה מסכמת לתואר שני

08/01/09

עבודה זו מציגה מתודולוגיה סדורה ליצירה של מודל Use Case מלא.

המתודולוגיה המוצגת בעבודה זו תוכל לשמש כמדריך למהנדס הדרישות בבואו לנתח את דרישות המערכת.

המתודולוגיה למעשה מאפשרת ליצור מודל של דרישות המערכת תוך שימוש שיטתי בטכניקת Use Cases. המתודולוגיה תומכת במידול איטרטיבי ובכך מאפשרת שימוש בה גם בפרויקטים גדולים ומורכבים. המתודולוגיה בנויה בתצורת מדריך עם שלבים ואף כוללת דוגמאות לתוצרים השונים. לכל שלב במתודולוגיה אותם אבני בנין וזה מאפשר עקומת לימוד פשוטה ומהירה.

המתודולוגיה מהווה אינטגרציה של מספר טכניקות, שיטות וכלים מתחומים שונים ומאפשרת להשיג בסופו של דבר את התוצרים הנדרשים.

המתודולוגיה הינה מונחית תוצרים – בכל שלב מצוין הרציונל , התוצרים הנדרשים בשלב זה , אילו פעילויות יש לבצע על מנת לקבלם וכוללת סעיפי הבטחת איכות המאפשרים למשתמש לבצע בקרת איכות על התוצרים השונים .

המתודולוגיה מסייעת למשתמש על ידי פירוט הכלים והטכניקות לקבלת התוצרים ומפרטת אילו סיכונים ואילו שגיאות נפוצות נקרות בדרכו להשגת התוצר הרצוי.

השילוב של מתן הרציונל בכל שלב עם האפשרות ליישום איטרטיבי מאפשרת "תפירה" של המתודולוגיה לכל ארגון ולכל פרויקט עם השינויים הנדרשים.

המתודולוגיה בנויה באופן לא קשיח ובכך מאפשרת גם את הרחבתה בעתיד בכיוונים שונים.

2. **מבוא**:

לאחרונה אנו עדים לכך כי יותר ויותר אנשים משתמשים בUse Cases על מנת לתאר תהליכים עסקיים ולתאר את דרישות והתנהגות המערכת. כתיבת Use Cases היא אמנות בפני עצמה אך גם כאשר הכותב מגיע לרמת מיומנות טובה במוקדם או במאוחר הוא ייתקל בשאלות רבות כגון: "כיצד עליי לבנות את מודל ה Use Case בצורה שיטתית?", "לאיזו רמת פירוט אני נדרש?", "מתי עליי לעצור?", "כיצד ניתן לוודא כי הUse Case איכותי?", "כיצד ניתן למדוד את איכות המודל כולו?", "אילו סיכונים קיימים בדרך?" וכו'.
המתודולוגיה המוצעת בעבודה זו באה לענות על שאלות אלו ורבות אחרות.
ע"פ מילון Merriam-Webster פירוש המילה *מתודולוגיה* הינו: "A series of related methods or techniques." מתודולוגיה היא פרוצדורה סיסטמטית המורכבת מסדרה של פעילויות ו/או טכניקות. מרבית הספרות המקצועית מתייחסת לנושא ה Use Cases כטכניקה עם מספר חוקי אצבע של "עשה" ו"אל-תעשה". עבודה זו מציגה מתודולגיה סדורה לבניית מודל Use Cases שלם ואיכותי.
העבודה בנויה בתצורה של פרקים אשר מהווים שלבים שונים של המתודולוגיה. כל פרק של המתודולוגיה מכיל תתי-שלבים המתארים את התוצרים, הפעילויות הדרושות לצורך השגתם ומה עוזר לבצע זאת.

<u>מהו מודל ה Use Case</u>?

מודל ה Use Case הוא מודל מתחום הנדסת תוכנה אשר מטרתו העיקרית לתאר את הדרישות הפונקציונליות של המערכת המפותחת. הבנה מלאה של דרישות המשתמשים והלקוחות היא תנאי הכרחי להצלחה של פרויקט פיתוח תוכנה (בקריטריונים של עלות/תועלת). איכות הדרישות היא גורם קריטי בהצלחה או כשלון של פרויקטי תוכנה. העקרון הבסיסי העומד מאחורי מודל ה Use Case הינו פשוט: אם ברצוננו להבין מה על המערכת לעשות באמת, עלינו להתמקד במי (או מה) הולך להשתמש בה או להיות מופעל על ידה. אחרי שנצליח לעשות זאת נגזור מכך את הפעילויות שהמערכת חייבת לבצע עבור אותם "משתמשים" בכדי להביא לתוצאה הרצויה.

מטרות העבודה :

ישנן וריאציות רבות של מודלים אשר עושים שימוש בטכניקת ה Use Case ,המודל הכללי של Cockburn הינו מוכוון מטרות משתמש וניתן להחילו בצורות שונות (מידול התהליך העסקי , מיצוי דרישות מערכת חדשה , תיעוד דרישות מערכת קיימת ועוד).

המודל של Kurt Bittner מתמקד במחזור החיים של ה Use Case ומתאר אותו מנקודות מבט שונות : מפתחי התוכנה (כיצד מושפע ה Use Case מתהליך הפיתוח), כותבי ה Use Case (כיצד ה Use Case מתפתח במהלך תהליך הכתיבה) , עבודה בצוות (הפעילויות המתבצעות במהלך יצירת מודל ה Use Case וכיצד משפיעות על עבודת הצוות ועל האינדיבידואל).

בספרות המקצועית אנו נמצא לרוב , התייחסות אל Use Cases כאל טכניקה ולא כאל מתודולוגיה. מטרת עבודה זו להציג תהליך שיטתי המתווה את אופן בניית ה Use Cases כמתודולוגיה סדורה שלמה[1].

המתודולוגיה המוצגת בעבודה זו תוכל לשמש כמדריך למהנדס הדרישות בבואו לנתח את דרישות המערכת תוך שימוש במודל ה Use Case.

המתודולוגיה בעלת מבנה ברור ופשוט , כפי שיתואר בהמשך , מכילה מרכיבי איכות ובכך מתאפשר למיישם לוודא כי התהליך כולו מבוצע באופן מבוקר ואיכותי.

המתודולוגיה המוצגת מאגדת טכניקות Use Case שונות ומגוונות ממקורות רבים ומאפשרת להחיל טכניקות אלו בצורה שיטתית על פרויקטים מתחומים שונים בעלי היקף מגוון.

רציונל :

העקרון הבסיסי העומד מאחורי המתודולוגיה הינו מידול דרישות המערכות תוך שימוש שיטתי בטכניקת Use Cases באופן איטרטיבי המאפשר את החלת המתודולוגיה על פרויקטים מסדרי גודל שונים של מורכבות. המתודולוגיה בנויה בתצורה של מדריך עם מספר שלבים בעלי אותם אבני בנין בכל שלב כך שעקומת הלימוד פשוטה ומהירה. המתודולוגיה המוצעת בעצם מהווה אינטגרציה של מספר טכניקות, שיטות וכלים מתחומים שונים על מנת לקבל בסופו של דבר את התוצרים הנדרשים.

המתודולוגיה המוצעת הינה מונחית תוצרים – בכל שלב מצוין הרציונל , התוצרים הנדרשים בשלב זה , אילו פעילויות יש לבצע על מנת לקבלם וכיצד ניתן להבטיח את איכותם.

המתודולוגיה מסייעת למשתמש על ידי פירוט הכלים והטכניקות לקבלת התוצרים ומפרטת אילו סיכונים ואילו שגיאות נפוצות נקרות בדרכו להשגת התוצר הרצוי.

השילוב של מתן הרציונל בכל שלב עם האפשרות ליישום איטרטיבי מאפשרת "תפירה" של המתודולוגיה לכל ארגון ולכל פרויקט עם השינויים הנדרשים.

---

[1] נסיונות דומים להצגת מתודולוגיית Use Case הינם : תהליך RESCU ותהליך ICONIX כאשר האחרון שם דגש על המעבר מ Use Case לקוד תוכנה.

מבנה המתודולוגיה :

המתודולוגיה המוצעת בנויה ע"פ שלבים כך שלכל שלב ישנם את אותם המרכיבים הבסיסיים ("אבני בניין"). המרכיבים שנבחרו עבור מתודולוגיה זו הינם מרכיבים אשר נפוצים במתודולוגיות פיתוח תוכנה שונות (Cockburn and Highsmith, 2006). ישנם קשרים בין המרכיבים השונים כפי שניתן לראות באיור3 .

להלן פירוט "אבני הבנין" של המתודולוגיה :

- ***רציונל ומטרות*** : מרכיב זה בא לתאר את הרציונל של השלב הנוכחי. מה נרצה להשיג בתום השלב ומדוע. המטרות מתוות את התהליך כולו אשר מתבצע בכל שלב.
- ***תוצרים*** : אלו הפלטים של כל שלב במתודולוגיה הנוצרים כתוצאה מביצוע פעילויות המתודולוגיה ע"י בעלי התפקידים השונים. תוצר יכול להיות כזה ש"הולך לפח" (למשל כרטיסי CRC[2]) או שיכול להיות קבוע לאורך כל הפרוייקט (כגון קוד מקור או מדריך למשתמש). תוצר יכול להיות שלם (כזה שהסתיים ואין כוונה לבצע בו שינויים) או חלקי (כזה שמתוכננות עבורו פעילויות נוספות על מנת להשלימו). תוצר למסירה הינו תוצר אשר עובר מבדקי איכות וחוצה את הגבול הארגוני (למשל מועבר ללקוח).
- ***פעילויות*** : מרכיב זה מתאר את סדרת הפעולות שיש לבצע על מנת להגיע לתוצרים הרצויים של השלב הנוכחי. ישנם מתודולוגיות המתמקדות בתוצרים (כגון מתודולוגית RUP) וישנן מתודולוגיות אשר שמות את הדגש על הפעילויות שעל האנשים לבצע (כגון Extreme Programming). במסגרת העבודה הנוכחית המתודולוגיה המוצעת תתמקד בתוצרים ,תתאר את הפעילויות הנדרשות לשם כך. בעלי התפקידים המעורבים יוזכרו בתום כל שלב אולם המתודולוגיה פחות מתמקדת באלמנטים של תקשורת ושיתוף בין האנשים השונים בתהליך.
- ***קלטים*** : מרכיב זה מפרט את הקלטים השונים הדרושים לצורך השגת המטרות של השלב הנוכחי. הקלטים יכולים להיות מסוגים שונים, כגון : מסמכי מערכת , סיכומי ראיונות , קוד קיים וכו'. במקרים מסוימים תוצרים משלבים קודמים יהוו קלט לשלב הנוכחי.
- ***כלים וטכניקות*** : טכניקות אלו הפרוצדורות שעל האנשים לבצע על מנת להשיג את הנדרש. חלקם מוכוונים לאדם בודד (למשל כתיבת Use Case שלדי) וחלקם לקבוצת אנשים (למשל סיעור מוחות). כלים אלו האמצעים הטכניים שבאמצעותם ניתן לבצע את הפרוצדורות הללו.
- ***איכות*** : מרכיב זה בא לתת מדדים וכלים לבחינת איכות התוצרים. בעזרת מרכיב זה ניתן לבצע בקרה על התהליך כולו ועל תוצריו השונים בכל שלב.
- ***שגיאות נפוצות וסיכונים*** : מרכיב זה מתאר את ה"מלכודות" הטיפוסיות אשר יש להזהר ולהמנע מהן. מכיוון שUse Cases מהווים מסגרת סמי-פורמלית עבור יצירת מבנה לסיפורים הרי לא מן הנמנע שקיימים לא מעט מלכודות ואף יתכן שימוש שגוי בטכניקות המוצעות.

---

[2] Class-Responsibility-Collaboration

תיאור של השגיאות הנפוצות יסייע לכותב להימנע מהן. מרכיב זה כולל גם תיאור של הסיכונים הקיימים בכל שלב ובכך מסייע לבצע ניהול סיכונים של התהליך.

- *בעלי תפקידים*: אלו האנשים המועסקים על ידי הארגון אשר שותפים בביצוע המתודולוגיה. לאנשים הללו ישנם כישורים מסוימים בהתאם למטרת העסקתם ולתפקיד אליו הם מיועדים. יתכנו מצבים בו אותו אדם מבצע מספר תפקידים שונים ועשוי להופיע בכל פעם עם "כובע" שונה.

איכות המודל

מודל ה Use Cases בראש ובראשונה חושף ומשקף את דרישות המערכת ולכן עליו לענות על הקריטריונים עבור SRS (ע"פ המלצות IEEE-STD-830-1998):

- נכונות הדרישות
  - אוסף כל ה Use Cases צריך לשקף את היכולות הנדרשות מהמערכת ולתאר את התנהגותה במצבים השונים.
- שלמות הדרישות
  - המודל מכיל את כל ה Use Cases המשמעותיים במערכת.
  - המודל מכיל תגובות לקלטים שונים כולל תרחישים אלטנטיביים והסתעפויות.
  - ה Use Cases כולם מתוארים באותו האופן בהתאם לסטנדרט אחיד (תבנית אחידה).
  - Use Case המורכב מתוצרים שונים כגון טבלאות, דיאגרמות וכד' יסומן כיחידה אחת עם תגית (label).
- עקביות (consistency) הדרישות
  - אין שני Use Cases המכילים מידע הסותר זה את זה.
  - תיאור השחקן בטבלת השחקנים ובתוצרים השונים תתאים לתיאורו בתבנית ה Use Case ולהתנהגותו.
  - מטרת ה Use Case תתאים לתיאורו ולתרחיש ההצלחה הראשי.
  - תרחישי ההסתעפות יתאימו לתרחיש ההצלחה הראשי ממנו התפצלו.
  - הקשרים השונים המתוארים בדיאגרמות ה Use Case יתאימו למידע הרלוונטי בתבנית ה Use Case.
  - התנאים השונים בתבנית יתאימו למטרת ה Use Case ויהיו רלוונטיים לאירועי ה Use Case בו הם מופיעים.
- עקיבות (trace-ability) הדרישות
  - קיימת מטריצת עקיבות בין Use Cases לאלמנטים מנוהלים אחרים (כגון Test Cases).
- מתן עדיפויות לדרישות
  - המודל מכיל תעדוף בין ה Use Cases השונים.

- ישנו תעדוף בין סיפוק מטרות השחקנים השונים.

- <u>דרישות אינן דו-משמעיות</u>
  - לכל Use Case ישנו פירוש יחיד. יש להימנע מתרחישים לא ברורים ומעורפלים. התרחישים יכתבו בצורה חד משמעית, פשוטה וברורה לקורא.

- <u>הדרישות ניתנות לבדיקה</u>
  - מודל ה Use Case ניתן לאימות בהתאם לפעילויות האיכות המפורטות בכל שלב.
  - מומלץ להצמיד Test Case לכל Use Case ולתחזק מטריצת עקיבות מתאימה.

- <u>הדרישות ניתנות לשינוי בקלות יחסית (modifiable)</u>
  - שינוי בUse Case לא גורר שינוי מרחיק לכת בתיכון וניתן בקלות יחסית לדעת מהן ההשפעות של השינוי.

המתודולוגיה המוצגת בעבודה זו איננה מאופיינת בצורה פורמלית ואיננה כוללת מדדי הבטחת איכות כמותיים אלא <u>מדדים איכותיים בלבד</u>.
עם זאת , ישנם מדדי איכות מסוימים המוצגים בשלבים השונים של המתודולוגיה המוצעת שניתן בקלות יחסית לבצע עבורם טרנספורמציה ממדד איכותי למדד כמותי. טרנספורמציה זו מחייבת עבודה מקדימה של ייצוג המודל באופן פורמלי וזהו נושא שניתן להרחיבו כהמשך לעבודה זו.

מתודולוגיות ומודלים שונים העושים שימוש בUse Cases

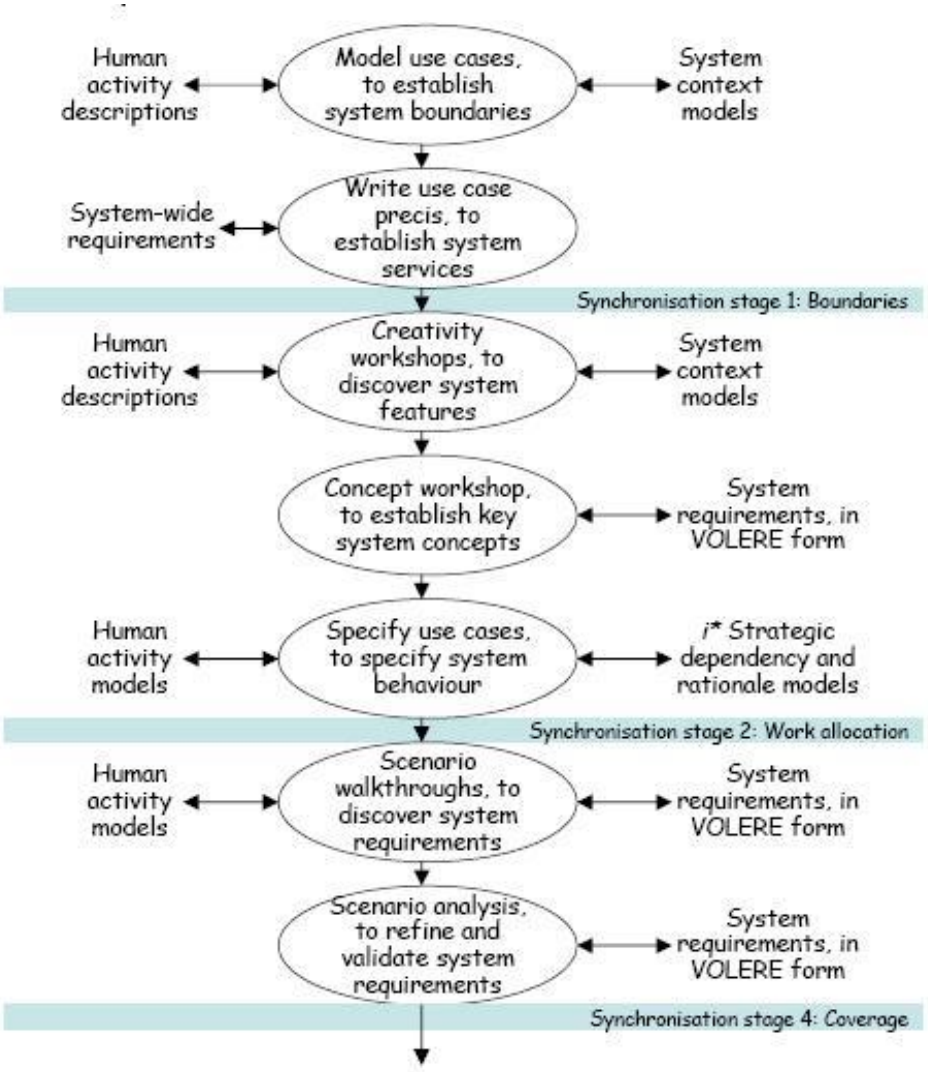
- מתודולוגיית RESCUE (Maiden and Robertson ,2005)

מתודולוגיה זו פורסמה בשנת 2003 ע״י פרופסור ניל מיידן מאוניברסיטת סיטי בלונדון. המתודולוגיה הינה חלק מטרנד של שילוב מספר טכניקות מעולם הנדסת הדרישות כך שנקודות החוזק של טכניקה אחת תפצה על נקודות החולשה של השנייה ולהיפך.

מתודולוגיית RESCUE משלבת בין מודל i* system goal לבין מודל human activity.

המתודולוגיה מיועדת עבור מערכות גדולות בהן נדרש בטחון רב בנכונות ושלמות ה Use Cases (ולכן גם הדרישות). המתודולוגיה נוסתה במסגרת הנדסת הדרישות של מערכת CORA-2 שהינה מערכת ממוחשבת גדולה הבאה לסייע לבקרי תעבורה אווירית.

להלן תרשים עקרוני לתהליך RESCUE :

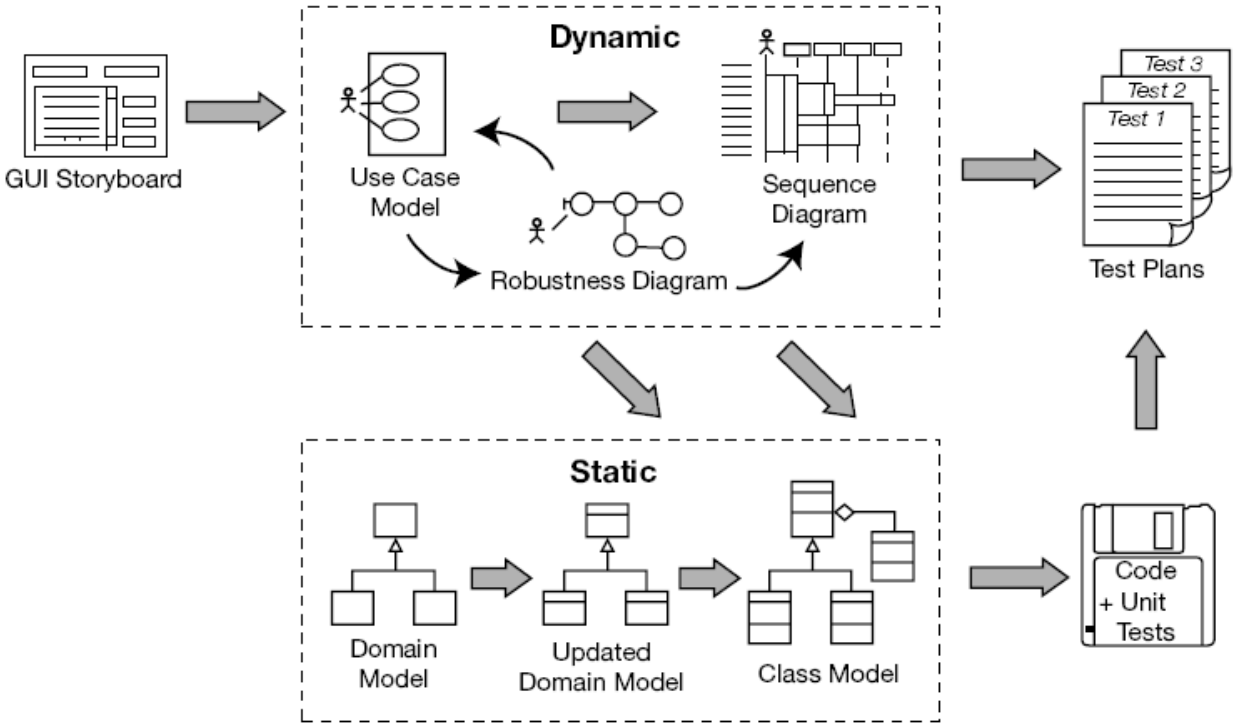


איור1

- <u>מתודולוגיית ICONIX</u> (Rosenberg & Stephens, 2005)

תהליך ICONIX הינו תהליך מינימליסטי אשר שם דגש על המעבר מ Use Cases לCode ונותן מענה לפער הקיים בין הניתוח לתכן. התהליך הינו חלק ממתודולוגיות agile. המתודולוגיה מתחלקת לתזרים דינמי ותזרים סטטי המבוצעים באופן איטרטיבי כאשר בכל איטרציה מבוצע חלק קטן של Use Cases (חבילה בודדת של Use Cases לא מפורטים). התזרים עובר דרך בדיקות יחידה ועד לקוד מקור. החוזק של התהליך הינו בפעילות הנקראת Robustness Analysis אשר מפחיתה את דו המשמעות בתיאור של Use Cases ובכך מבטיחה שהם נכתבים בקונטקסט המתאים לתחום שאותו הם ממדלים. התהליך גורם ל Use Cases להיות קלים לתיכון , בדיקה והערכה.

<u>להלן תרשים עקרוני לתהליך ICONIX</u> :



איור2

08/01/09

השוואה בין המתודולוגיות השונות

	RESCUE	ICONIX	המתודולוגיה המוצעת בעבודה זו
קונספט	מידול דרישות המערכת ע״י שילוב של הבנת הפעילויות הנעשות ע״י האנשים הקשורים למערכת ומידול של גבולות המערכת, השחקנים והמטרות.	תהליך מינימליסטי אשר עושה שימוש ב UC כחלק מהמודל אשר בסיומו מתקבל קוד תוכנה. מסווג כמתדולוגיית Agile. שימוש ב-4 סוגי דיאגרמות UML בלבד.	תהליך סדור ואיטרטיבי למידול המערכת הבנוי בצורת מדריך וכולל רשימת תוצרים בכל שלב כולל פעילויות להבטחת איכות התוצרים.
דגש עיקרי	אינטגרציה בין טכניקות שונות מעולם הנדסת הדרישות כאשר הגרעין הינו UC.	גישור על הפער בין תכן לקוד. שימוש בטכניקות agile ובתהליך איטרטיבי.	התאמה לפרויקטים בסדרי גודל שונים. כלים להבטחת איכות התוצרים באופן מובנה כחלק מהמתודולוגיה. ניהול סיכונים והימנעות משגיאות נפוצות.
יתרונות עיקריים	תמיכה במערכות גדולות ומורכבות. תהליך מובנה ומקיף הכולל טכניקות רבות.	תהליך ״קל״ (lightweight) , תוך 4 שלבים בלבד מגיעים מדרישות לקוד תוכנה.	המתודולוגיה מתאימה לשימוש גם עבור מערכות גדולות וגם עבור מערכות בסדרי גודל קטנים יותר. בנויה בתצורת מדריך - עקומת לימוד מהירה.
חסרונות עיקריים	תהליך מורכב , עקומת לימוד גבוהה. שימוש במודל לא נפוץ בשם i* system goal.	לא מתאים למידול מערכות גדולות ומסובכות.	המידול מסתיים בתוצרי ה UC ואיננו ממשיך עד לקוד תוכנה. אין עדיין מספיק נסיון מעשי במתודולוגיה המוצעת בעבודה זו.

טבלה1

בטבלה הנ״ל השווינו את המתודולוגיה המוצעת ל2 מתודולוגיות אחרות המייצגות 2 גישות שונות, כאשר הראשונה (RESCUE) הינה מתדולוגיה ״כבדה״ ופורמלית והשניה (ICONIX) הינה ״קלה״ ומגיעה מעולם ה Agile.

כפי שניתן לראות מהטבלה הנ״ל המתודולוגיה המוצעת בעבודה זו מגשרת על החסרונות העיקריים הקיימים במתודולוגיות RESCUE ו ICONIX.

נקודת החוזק של המתודולוגיה הינה בפשטותה. כמובן שיתכנו קשיים ביישומה וככל שהמשתמש מיומן יותר בטכניקת Use Cases כך יהיה לו קל יותר אך הנקודה החשובה היא שהתהליך עצמו איננו מורכב. המבנה של המתודולוגיה מקל על המשתמש ביישומה.

התהליך עצמו איננו "קל" או "כבד" באופן מובהק וניתן ליישום גם במערכות גדולות וגם עבור מערכות קטנות. יתרון נוסף של המתודולוגיה המוצעת הינו ההתייחסות המובנית בתוך התהליך לנושא של הבטחת איכות התוצרים וניהול הסיכונים. במתודולוגיות המוכרות האחרות נושאים אלו אינם חלק מובנה בתהליך.

חסרונות המתודולוגיה נעוצים באופן שילובה בתהליך פיתוח הפרוייקט כולו. בניגוד למתודולוגיית ICONIX בה מוסבר כיצד מגיעים בסוף לקוד תוכנה במתודולוגיה זו אנו מסיימים עם תוצר של חבילות UC המהוות מידול של דרישות המערכת. בנושא זה יש להעמיק ולחקור כיצד ניתן לשלבה באופן מיטבי כך שניתן יהיה להפיק SRS ואף תכן חלקי מתוך תוצרי המתודולוגיה.

מגבלה נוספת של המתודולוגיה על פני מתודולוגיית RESCUE הינה בחוסר במדדים כמותיים. מדדי הבטחת איכות המודל הינם איכותיים ולא כמותיים ובכך מקשים לבצע השוואה בין מידול של פרויקטים שונים.

אולם , המתודולוגיה המוצעת איננה קשיחה וניתנת להרחבה בצורה קלה יחסית. לדוגמא, ניתן להוסיף מדד כמותי להערכת המאמץ הנדרש לפיתוח הפרויקט ולשלבו בסעיפי הבטחת האיכות בשלבים השונים עם התייחסות לאיטרציה המבוצעת. סיכום התוצאות בכל שלב ושלב תתן הערכה כוללת של המאמץ הנדרש עבור פיתוח הפרויקט כולו. המדד המומלץ לשימוש הינו הרחבה של מדד Use Case Points התומך במתודולוגיות אינקרמנטליות עבור פרוייקטים מורכבים (Mohagheghi, P., Anda, B. and Conradi, R., 2005).

עבודה מסכמת לתואר שני

08/01/09

השלבים השונים של המתודולוגיה

לכל שלב של המתודולוגיה מוקדש פרק נפרד.
כל פרק מכיל מספר תתי-שלבים אשר בכל אחד מהם מתוארות הפעילויות לצורך השגת התוצר הנדרש של תת שלב זה. כל שלב מכיל את אותם מרכיבים יסודיים של המתודולוגיה כפי מתואר בהמשך.

להלן פירוט השלבים השונים:

שלב I - תיחום גבולות המערכת.

שלב II - זיהוי שחקנים ומטרות.

שלב III - גיבוש Use Case שלדי.

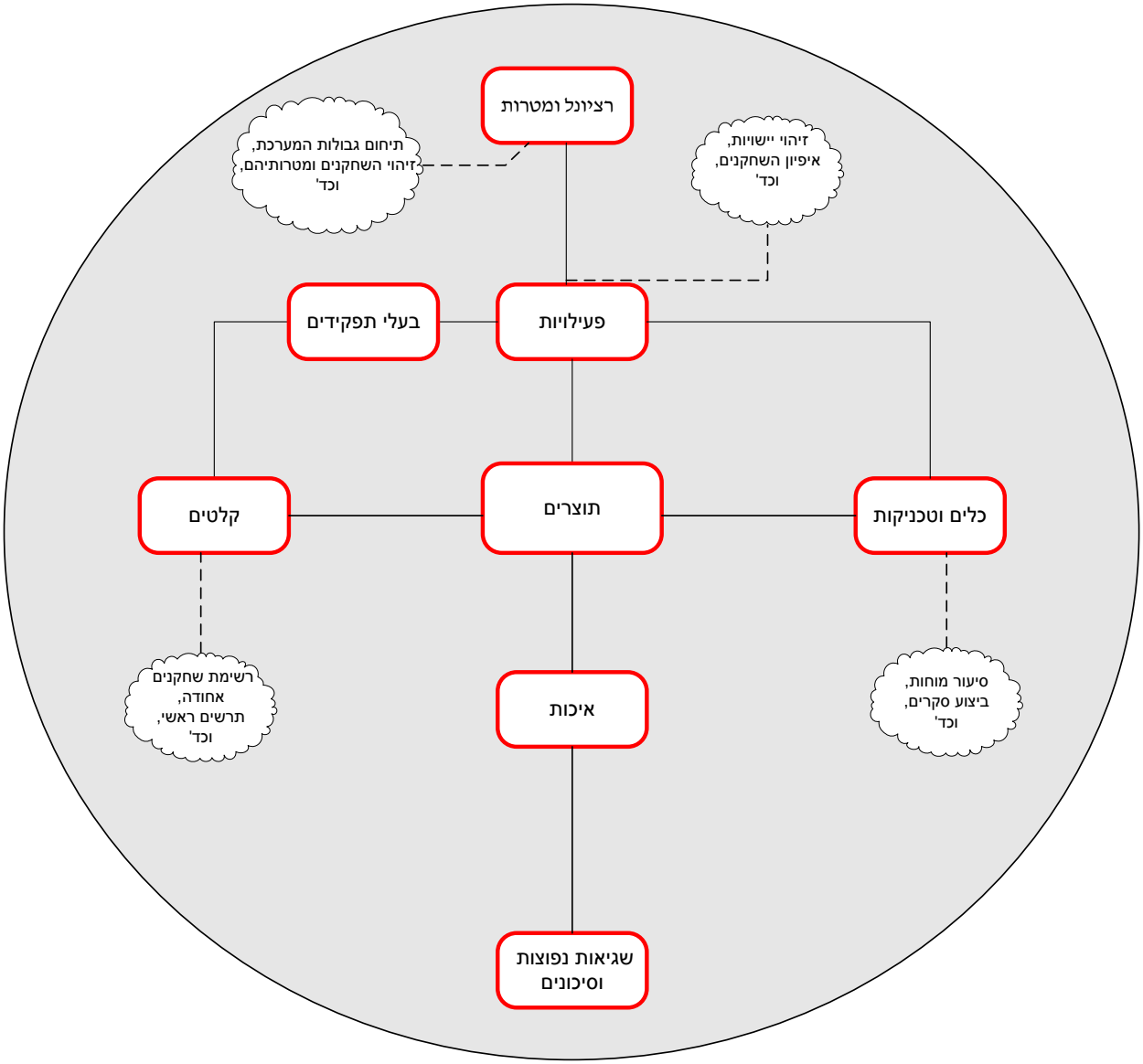
שלב IV - הרחבת ה Use Case.

שלב V – טיפול בהרחבות.

שלב רוחבי - ניהול ה Use Cases והבטחת איכות.

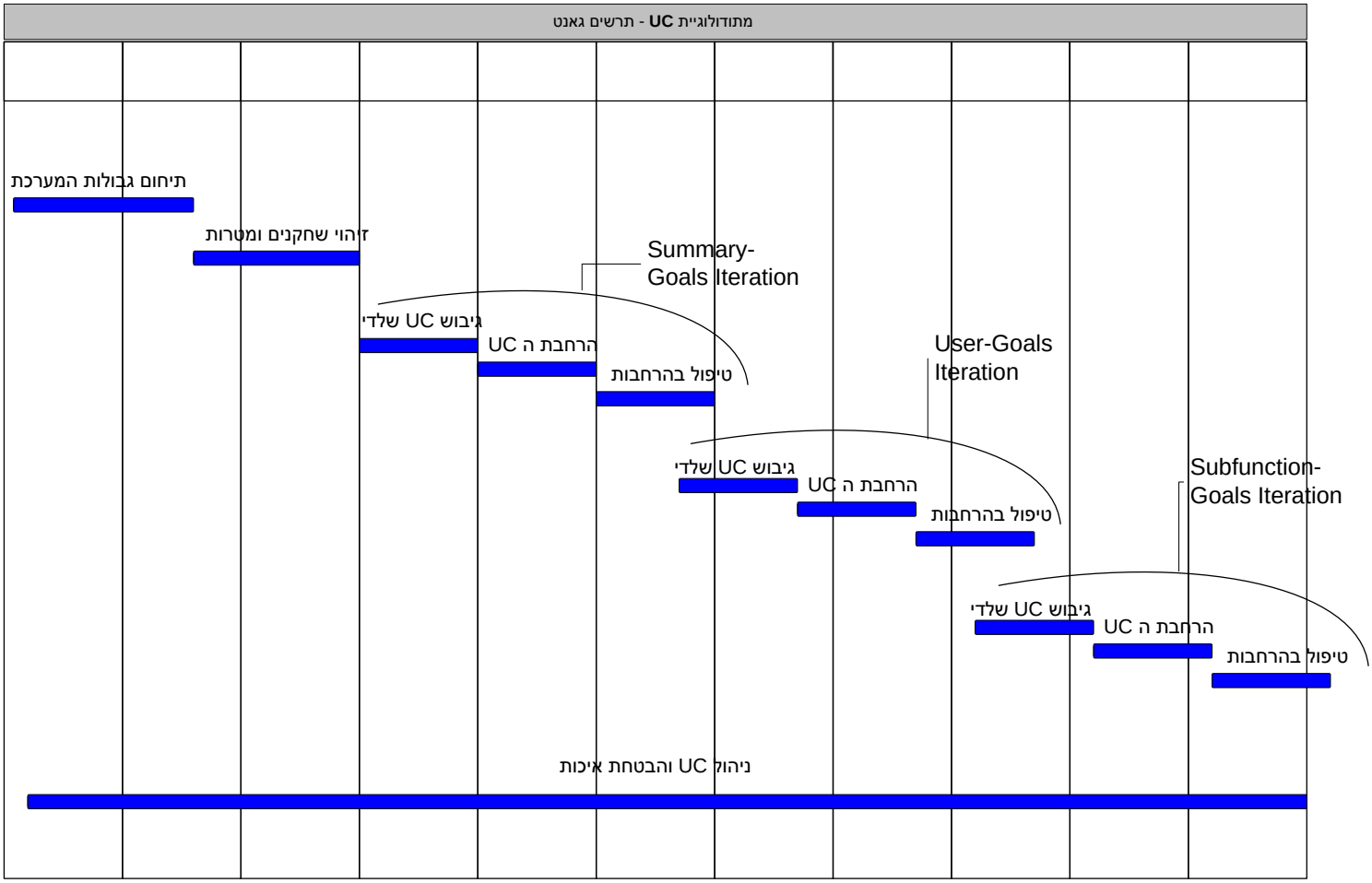
08/01/09

מרכיבי מתודולוגיית ה Use Case Model המוצעת:



איור3

תרשים גאנט עקרוני לביצוע השלבים השונים של המתדולוגיה המוצעת [3] :



איור4

יחידות הזמן בתרשים תלויה במורכבות הפרויקט.

בעבור פרויקט מסדר גודל קטן עד בינוני כל עמודה צפויה לארוך בין 3 ימים לשבוע.

בעבור פרויקט בעל סיבוביות גבוהה עד גבוהה מאד נצפה ליחידות זמן של בין שבוע לחודש.

[3] ניהול המודל ופעילויות הבטחת האיכות מתבצעות לאורך כל השלבים כמפורט בהמשך בכל שלב ושלב.

3. שלב I - תיחום גבולות המערכת:

<u>רציונל</u>:

קביעת גבולות המערכת צריכה להתבצע בשלבים הראשוניים של הפרוייקט מכיוון שהיא משפיעה באופן חזק על הגורמים העיקריים בפרוייקט : עלות , תכולות , לוח זמנים , משאבים וכד'. במידה ונעשתה טעות בשלב זה תיקונה יהיה יקר יותר ככל שתתגלה בשלב מאוחר יותר. המטרה העיקרית בשלב זה היא לקבוע אילו רכיבים הם בתוך המערכת המפותחת ואילו רכיבים מחוצה לה. רכיבים אשר נמצאים מחוץ למערכת ואינם מפותחים במסגרת המערכת משפיעים על המערכת (במידה וקיימת אינטראקציה מולם יש צורך לפתח ממשק). במונחים של UML יש להגדיר עבורם Boundary Class- מחלקה אשר תפקידה לתקשר בין ישות בתוך המערכת לישות אשר חיצונית למערכת. Cockburn (2000) מתייחס לשלב זה כאל שלב קביעת ה Scope הפונקציונלי. פעילות של תיחום גבולות המערכת אשר נראית טריוויאלית במבט ראשון מתגלית כמשימה כלל לא פשוטה. כחלק מקביעת גבולות המערכת מקובל גם לבצע Scoping למערכת – חלוקת הזמן והמשאבים לרכיבים השונים אשר יפותחו במסגרת המערכת (Schneider & Winters, 2005).

1.3. תוצר נדרש: "הצהרת הבעיה" (תוצר שלם)

מסמך הצהרת הבעיה הינו מסמך המכיל תיאור של הבעיה שהמוצר/המערכת באה לפתור עם מיקוד על הבעיה ומהי התועלת בפתרונה.

1.1.3. המטרה:

מטרת תת שלב זה הינה לבצע ניסיון ראשוני להבין "מי נגד מי?" – מהי הבעיה שהמערכת באה לפתור? ממה מורכבת קהילת בעלי העניין? מהם הצרכים של בעלי העניין וכיצד המערכת תספק אותם?

2.1.3. פעילויות:

- ❖ סקירת מסמכים ונכסי ידע

בפעילות זו ננסה להבין איזו עבודה כבר נעשתה – אילו מסמכים קיימים ? אילו פעילויות התבצעו? אילו חלופות כבר נשקלו? וכד'..

פעילות זו מחייבת ביצוע של מעבר על מסמכים , תוצרים , אימיילים , Memos וכד'. בעצם ביצוע פעילות זו אנו משתמשים באסטרטגיה של Reuse כבר בשלב הדרישות. להלן מספר שאלות אשר נרצה למצוא מידע עבורם:

- **?** האם קיים מסמך חזון למערכת?
- **?** מי איננו מעוניין במערכת החדשה? מי מעוניין בבניית המערכת ?
- **?** האם נשקל שימוש בחבילות קיימות? (COTS[4])
- **?** אילו חלופות הועלו ומהן החלופות שכבר נפסלו?
- **?** מהי מידת המעורבות של ההנהלה הבכירה בניהול פרוייקט זה?
- **?** האם כבר קיימת הצעה מובנית של פתרון לבעיה? האם הפתרון המוצע מורכב מתוכנה בלבד? חומרה? שילוב של חומרה ותוכנה?

- ❖ כתיבת "הצהרת הבעיה"

במידה ובפעילות הקודמת זיהינו כי קיים מסמך חזון למערכת (לחילופין קיים חזון אך הוא איננו מרוכז במסמך יחיד) יש לדלות את הצהרת הבעיה מהמסמכ/ים. ייתכן מצב כי קיים חזון למערכת אך הוא איננו מתועד. במקרה זה יש לבצע בדיקה אל מול גורמי מפתח כגון ההנהלה הבכירה ולקבל תיאור של החזון. לעיתים תחילת הפעילות של מידול Use Case תגרום להנעה של תהליך אשר בסופו יווצר מסמך חזון המערכת. את מסמך הצהרת הבעיה יש לאשר אצל ההנהלה הבכירה ולוודא כי הוא אכן מתאים לחזון המערכת (Bittner & Spence, 2002).

---

[4] Commercial off-the-shelf

3.1.3. <u>קלטים</u>:

❖ <u>מסמך חזון המערכת</u> (Vision Document)

מסמך חזון המערכת מכיל את הצרכים והתכונות העיקריות הנדרשות מהמערכת. המסמך מתאר את החזון של בעלי העניין מהמערכת המפותחת ומהווה בסיס לחוזה בין הלקוח לארגון המפתח. במסמך החזון ניתן לראות מה הניע את יוזמי הפרוייקט ,כיצד הם רואים את התפתחות המערכת בעתיד ומהן דרישות העל מהמערכת. מסמך החזון הוא מסמך אשר לאורו מפותחת המערכת. מסמך החזון נכתב מנקודת מבטו של הלקוח ומתמקד בתכונות העיקריות והמהותיות של המערכת. לרוב, המסמך יתייחס גם לאיכות המערכת הנדרשת. מסמך החזון צריך להכיל גם תיאור של תכונות מערכת אשר נשקלו אך הוחלט לבסוף שלא להכניסם. מסמך החזון צריך להכיל קיבולת נדרשת (נפחים, זמני תגובה, דיוקים וכו') , פרופיל משתמשים וממשקים עם יישויות הנמצאות מחוץ לגבול המערכת (Bittner & Spence, 2002). לעיתים רבות מסמך החזון כללי מדיי וזאת מכיוון שהלקוח בעצמו עדיין איננו יודע מהי המערכת הסופית בה הינו מעוניין. מסמך חזון המערכת מהווה אמצעי עיקרי לתקשורת בין ההנהלה , השיווק וצוותי הפרוייקט (Kruchten, 2003).

להלן הפרקים אשר יכללו במסמך החזון :

- מיקום (Positioning):
  - מהי ההזדמנות העסקית שהמוצר בא לנצל?
  - הצהרת הבעיה – תיאור של הבעיה שהמוצר בא לפתור עם מיקוד על הבעיה ומהי התועלת בפתרונה.
  - שיווק – תיאור כוחות השוק השונים המניעים את ההחלטות לגביי המוצר.
  - סביבת הלקוח – מהן סביבות הלקוח בהן ניתן ליישם את המוצר.
- בעלי עניין ומשתמשים :
  - בעלי עניין וצרכי המשתמשים – תיאור צרכים עיקריים של בעלי העניין והמשתמשים אשר המוצר בא לספק. התיאור כללי ונותן רקע לדרישות ומדוע הן אכן מוצדקות.
  - סקירת על של המוצר – בחלק זה יתוארו ברמת-על יכולות המערכת, ההנחות, התלויות ואלטרנטיבות לפיתוח המוצר.
  - תכונות – בחלק זה תסופק רשימת תכונות של המוצר. תכונות הן יכולות ברמת-על הנדרשות מהמערכת על מנת לספק את התועלת ההכרחית למשתמשים ואת הצרכים של בעלי העניין. זהו החלק הארוך ביותר בדרך כלל במסמך החזון.

08/01/09

- דרישות נוספות – בחלק זה תסופק רשימה של דרישות על נוספות אשר אינן נכללות כתכונות המוצר. ברשימה זו יופיעו אילוצים ודרישות מהסביבה שבה יופעל המוצר.

❖ <u>מסמכים אחרים ונכסי ידע</u>

זהו אוסף המסמכים ושאר נכסי ידע הקיימים בארגון בהקשר של המערכת המפותחת. במידה ואנו דנים בשדרוג של מערכת קיימת סביר להניח כי ישנם מסמכים רבים המתארים את המערכת הקיימת. נכסי ידע אחרים יכולים להיות למשל פורטל פנימי בארגון המרכז את הפעילויות והמסמכים של הפרוייקט.

4.1.3. <u>כלים וטכניקות</u>:

❖ Reuse - שימוש בנכסי ידע קיימים בארגון.

5.1.3. <u>איכות</u>:

❖ להלן קווים מנחים לבדיקת איכות של התוצר :

- קיים תיאור של הבעיה
- קיים תיאור של הגורמים המושפעים מהבעיה
- קיים תיאור של השפעות הבעיה
- קיים תיאור של הפתרון
- קיים תיאור של התועלת שתנבע מפתרון מוצלח
- אין פירוט רב מדיי של פרטים
- הצהרת הבעיה לא אורכת יותר מדף אחד

6.1.3. <u>סיכונים ושגיאות נפוצות</u>:

❖ הסיכון העיקרי הינו התמקדות בבעיה משנית ולא בבעיה העיקרית אותה המערכת באה לפתור. במקרה זה ישנה סבירות גבוהה לשרשרת של טעויות אשר בסופה יתקבלו גבולות מערכת שגויים והמודל יכיל שגיאות רבות עקב כך.

❖ <u>שגיאות נפוצות ודרכי התמודדות</u> (Savransky, 2000) :

השגיאה	**תיאור השגיאה**	**טכניקת מניעה**
תיאור כללי מדי	הצהרת הבעיה כללית מדיי	הפוך את הבעיה למוחשית יותר, תאר אותה בקונטקסט של סיטואציה ספציפית.
תיאור צר מדיי	הצהרת הבעיה ברורה רק לאלו שכתבו אותה בצורה פורמלית	הסבר את הבעיה במילים פשוטות. תאר אותה כך שאדם ללא רקע מוקדם יוכל להבינה.

בעיה שגויה	פתרונות הבעיה לא משפיעים בצורה חיובית כמתבקש	נסה להבין את התוצאה שתתרחש בעקבות פתרון הבעיה
	נעשה נסיון לפתור בעיה הרבה יותר מורכבת מזו שבאמת צריך לפתור	צור סיטואציה חדשה ובחר בעיה שפתרונה מספק את התוצאה הרצויה.
	תיאור קצר של הבעיה לא לוקח בחשבון וראיציות רבות נוספות.	בצע מיצוי מחודש של הבעיה והתחשב בגורמים של זמן הדרוש לפתרון רחב של הבעיה.
	הפתרון המוצע לבעיה איננו פותר תת-בעיה מסויימת כפי שפותר את יתר תת הבעיות.	בחן את מבנה הבעיה וסווג את כל תת הבעיות האלמנטריות. הצג פתרון נפרד לתת הבעיות האלמנטריות.
גישה לא שיטתית	ההצהרה מתארת רק את הבעיה הברורה. כאשר הפתרון לבעיה מוצג ניתן להבין שהבעיה היא רק חוליה בשרשרת של בעיות.	בצע מיצוי של שרשרת הבעיות כולה ומצא את המפתח המשותף לכולן.

טבלה2

7.1.3. <u>דוגמא לתוצר הצהרת הבעיה</u>:

"משיכת מזומנים הינו תהליך ידני המחייב אחזקה שוטפת של טלרים ומתן שירות לאורך שעות רבות בסניפים השונים. התהליך מחייב את הלקוחות בהמתנה ממושכת בתורים וכל זאת על מנת לבצע פעולה פשוטה של משיכת מזומנים. תהליך זה גורם לאי שביעות רצון מצד הלקוחות ולהוצאות שוטפות גדולות מצד הבנק. מערכת ה ATM באה לספק פתרון יעיל לבעיה. המערכת החדשה תורכב ממכונות טלר אוטומטיות (ATM) אשר ישותפו בין בנקים שונים בקונצריום באמצעות רשת בנקאית מאובטחת. כל בנק יספק את המחשב המרכזי שלו לצורך תחזוקת החשבונות וביצוע הטרנזקציות הנדרשות מולו. מכונות ה ATM יספקו למשתמשים מכל בנק שהוא שירותים שונים כאשר השירות העיקרי הינו משיכת כסף מזומן. אמצעי זיהוי הלקוח יבוצע באמצעות כרטיס מגנטי בנקאי. הלקוחות יחוייבו בדמי שימוש בעבור ביצוע פעולות באמצעות מכונות ה ATM. מכונות ה ATM הינן חלק מתהליך כולל של ייעול הארגון אשר מטרתו לקצץ בשעות פעילות הסניפים ובכח אדם".

2.3. תוצר נדרש: רשימת יישויות (תוצר חלקי)

רשימת הישויות הינה רשימה של כל היישויות אשר מתקשרות למערכת. רשימה זו עשויה לכלול גם יישויות אשר הינן חלק מהמערכת. הרשימה עשויה להכיל כפילויות של אותה יישות המתוארת באופנים שונים.

1.2.3. המטרה:

מטרת תת-שלב זה הינה להכין תשתית לצורך פעילות שתתבצע בהמשך ובה נזהה את השחקנים במערכת. בשלב זה אנו עדיין לא מזהים שחקנים אף על פי שיתכן כי יישויות רבות אשר נזהה כעת ימצאו בהמשך כשחקנים במודל.

2.2.3. פעילויות:

- ❖ זיהוי יישויות

בפעילות זו נכין רשימה של יישויות אותן נזהה ע"י מעבר על מסמכי המערכת. בשלב זה אין לטפל בכפילויות. נרצה לאסוף כמה שיותר יישויות שנזהה הקשורות למערכת המפותחת. כל משתמש או מערכת שנגלה נסמן כיישות. לאחר שיש בידינו רשימה ארוכה של יישויות מומלץ לבצע סיעור מוחות בהשתתפות בעלי תפקידים נוספים ולהעלות לדיון את השאלה (עבור כל יישות) האם היישות חיצונית למערכת או שהינה חלק מהמערכת המפותחת? בתום הפעילות תהיה בידנו רשימת יישויות אשר מסווגות ל In ו Out (Cockburn, 2000).

3.2.3. קלטים:

- ❖ מסמכי מערכת (מסמך חזון , מסמכי דרישות , תכתובות שונות , תרשימי מערכת וכו').

4.2.3. כלים וטכניקות:

- ❖ Reuse - שימוש בנכסי ידע קיימים בארגון.
- ❖ שימוש ברשימת In/Out – לצורך סיווג היישויות.
- ❖ סיעור מוחות – לצורך קבלת החלטה בנוגע לסיווג הישויות שזוהו.

5.2.3. איכות:

- ❖ להלן קווים מנחים לבדיקת איכות של התוצר:
  - כל היישויות אשר זוהו מתוך הקלטים השונים מופיעות בטבלה
  - על שם היישות להיות עקבי עם הופעתה בחלקים אחרים (כגון בתרשימים).
  - העדף לכלול יישויות מיותרות מאשר להחסיר.

6.2.3. <u>סיכונים ושגיאות נפוצות</u>:

- הסיכון העיקרי בתת-שלב זה הינו חוסר זיהוי של יישויות עיקריות המתקשרות עם המערכת או סיווג שגוי של ישויות עיקריות. טעות זו עשויה להוביל לקביעת גבולות מערכת שגויים.
- להלן שגיאות נפוצות ודרכי התמודדות:
  - סיווג יישות שאיננה חלק מהמערכת פנימה (In במקום Out).
  - סיווג יישות שהינה חלק מהמערכת כחיצונית (Out במקום In).

במקרה הראשון , השגיאה תגרום לנו לעבודה רבה ומיותרת וכן פספוס של ממשק אשר צריך לזהות ולטפל בו.

במקרה השני, השגיאה תגרום ליצירת ממשק מיותר ומחלקת גבול מיותרת.

בשני המקרים , ככל שהשגיאה תתגלה מאוחר יותר כך יהיה קשה יותר לתקנה ותיקונה "יעלה ביוקר".

דרכי התמודדות:

- ביצוע סיעור מוחות.
- ביצוע סקר חוזר על רשימת הישויות (יבוצע בשלב של זיהוי השחקנים והמטרות). ככל שהפיתוח מתקדם התמונה הולכת ומתבהרת.האופן בו אנו מבינים את התנהגות המערכת הולך ומשתפר ולכן בשלב הבא יש לחזור לאחור ולבצע בקרה על תוצר חלקי זה.

7.2.3. <u>דוגמא לתוצר רשימת היישויות</u>:

Entity	**In**	**Out**
Customer		✔
Technician		✔
Mainframe		✔
Logger	✔	
Teller		✔
Loan Officer		✔
Communicator	✔	
Credit Card Company		✔
Security System		✔
Printer	✔	
Math Engine	✔	

טבלה3

8.2.3. <u>השלמת התוצר</u>:

יש לעדן את הרשימה ע״י מעבר מחודש בעת הגעה לשלב הבא במתודולוגיה שהינו זיהוי שחקנים ומטרות.

3.3. תוצר נדרש: סקופ פונקציונלי (תוצר חלקי)

סקופ זה מתייחס לשירותים שהמערכת מספקת אשר בסופו של דבר ישוקפו בעזרת ה Use Cases. כאשר אנו בתחילת הפרוייקט אנו מחליטים על הסקופ הפונקציונלי במקביל לתהליך זיהוי ה-Use Cases. ישנה השפעה הדדית בין שני התהליכים (Cockburn, 2000).

1.3.3. המטרה:

מטרת תת-שלב זה לאתר את הפונקציונליות העיקרית אשר נדרשת מהמערכת. התוצר הנדרש הינו חלקי ויכיל את הפונקציות העיקריות אשר יתן לזהות כבר בתחילת הפרוייקט. התוצר יושלם בהמשך כאשר נגיע לשלבים הבאים במתודולוגיה.

2.3.3. פעילויות:

- ❖ זיהוי פונקציות עיקריות הנדרשות מהמערכת

בפעילות זו נכין רשימה של תכונות המערכת הנדרשות אותן ניתן לאתר ע"י מעבר על מסמכי המערכת. לאחר שיש בידינו רשימה ארוכה של פונקציות נדרשות מומלץ לבצע סיעור מוחות בהשתתפות בעלי תפקידים נוספים ולהעלות לדיון את השאלה (עבור כל פונקציה) האם הפונקציה המדוברת היא חלק מהמערכת המפותחת או שהינה חיצונית ואיננה חלק מדרישות המערכת? בתום הפעילות תהיה בידנו רשימת פונקציות אשר מסווגות ל In ו Out.

3.3.3. קלטים:

- ❖ מסמכי מערכת (מסמך חזון , מסמכי דרישות , תכתובות שונות , תרשימי מערכת וכו').

4.3.3. כלים וטכניקות:

- ❖ Reuse - שימוש בנכסי ידע קיימים בארגון.
- ❖ שימוש ברשימת In/Out – לצורך סיווג הפונקציות.
- ❖ סיעור מוחות – לצורך קבלת החלטה בנוגע לסיווג הפונקציות שזוהו.

5.3.3. <u>איכות</u>:

❖ להלן קווים מנחים לבדיקת איכות של התוצר:

- כל תכונות המערכת אשר זוהו מתוך הצהרת הבעיה ומסמך חזון המערכת מופיעות בטבלה
- בכל שורה בטבלה תסומן אפשרות In במידה והפונקציונליות הינה חלק מהמערכת המפותחת.
- בכל שורה בטבלה תסומן אפשרות Out במידה והפונקציונליות איננה חלק מהמערכת המפותחת.

6.3.3. <u>סיכונים ושגיאות נפוצות</u>:

❖ הסיכון העיקרי בתת-שלב זה הינו קביעת סקופ פונקציונלי שגוי בעקבות חוסר זיהוי של פונקציה עיקרית במערכת. במידה ונכניס פנימה פונקציה מיותרת נגרום לעבודה רבה ללא צורך אך הנזק יהיה קטן יותר מאשר במקרה בו נפספס פונקציה עיקרית ומהותית הנדרשת מהמערכת המפותחת.

❖ להלן שגיאות נפוצות ודרכי התמודדות:

- הכנסת פונקציה שאיננה חלק מהמערכת פנימה (In במקום Out).
- סיווג פונקציה שהינה חלק מהמערכת כחיצונית (Out במקום In).

במקרה הראשון , השגיאה תגרום לנו לעבודה רבה ומיותרת ופספוס של ממשק אשר צריך לזהות ולטפל בו.

במקרה השני, השגיאה תגרום לפספוס של פונקציה/תכונה מהותית של המערכת וככל ששגיאה זו תתגלה מאוחר יותר כך יהיה קשה יותר לתקנה ותיקונה יצריך שינויים רבים.

דרכי התמודדות:

- ביצוע סיעור מוחות.
- ביצוע סקר חוזר על הסקופ הפונקציונלי (רשימת תכונות) אשר יבוצע בשלב של זיהוי השחקנים והמטרות.

7.3.3. <u>דוגמא לתוצר הסקופ הפונקציונלי</u> :

להלן רשימת In/Out עבור מערכת כספומט בבנק. הפונקציות העיקריות הנדרשות ממערכת ה ATM הינן אלו המסווגות כ In :

Topic	**In**	**Out**
Deposit Money		Out
Withdraw Money	In	
Maintain Machine	In	
Check Balance	In	
CreditCard Company Computer System		Out
Printer	In	
Bank Mainframe		Out

טבלה4

8.3.3.<u>השלמת התוצר</u> :

יש להשלים את הרשימה בשלב הבא במתודולוגיה שהינו זיהוי שחקנים ומטרות.

4.3. תוצר נדרש: תרשים Use Case / Context מערכתי (תוצר מלא)

Context Diagram[5] זה תרשים הלקוח משיטות ניתוח מבני שפותחו כבר ב-1970. למרות הגיל המופלג של שיטה זו, דיאגרמות תוכן עדיין נמצאות בשימוש בלא מעט ארגונים וזאת מכיוון שהן משקפות בצורה טובה את הסביבה בה שוכנת התוכנה (Wiegers, 2006).

Use Case Diagram הינה דיאגרמה תקנית בשפת UML אשר הפכה לטכניקה חזקה ונפוצה בתחום של איסוף דרישות הלקוח (Object Management Group ,1999).

דיאגרמת Use Case איננה באה במקום Use Cases שהינם "יצורים" טקסטואליים במהותם.

בדיאגרמת תוכן אנו מקבלים גם את הקונטקסט של המערכת אל מול העולם החיצוני כאשר בדיאגרמת Use Case הדגש הוא על התנהגות המערכת.

המשותף ל-2 הדיאגרמות הינו בייצוג ברור של גבולות המערכת, הישויות המשתתפות והפונקציונאליות העיקרית אותה מספקת המערכת וכל זאת בתצוגה גרפית ברמת על.

1.4.3. <u>המטרה</u>:

מטרתו של תת-שלב זה הינה להציג בתוצר אחד בצורה ברורה גם את גבולות המערכת וגם את הפונקציות העיקריות הנדרשות מהמערכת. התרשים מרכז את כל המידע שנאסף עד כה במסמכים וברשימות שיצרנו לעיל ומציג זאת בצורה גרפית. תוצר זה הוא מעין הסכם על גבולות המערכת ותכולותיה וניתן להציגו גם ללקוח וגם לצוותי הפיתוח. התרשים הינו מערכתי ואיננו מפרט תכונות משניות או פונקציות שאינן ברמת על.

2.4.3. <u>פעילויות</u>:

- <u>יצירת דיאגרמת תוכן / דיאגרמת Use Case_ מערכתית</u>.

זוהי הפעילות המסכמת שנבצע בשלב תיחום גבולות המערכת. לאחר שאספנו מספיק מידע על הבעיה , המערכת המוצעת והישויות המשתתפות – ניתן לבנות תרשים Use Case ראשוני אשר בעזרתו ניתן להציג באופן ברור את גבולות המערכת. תרשים זה מהווה קו בסיס טוב להתחלת המודל.

- במידה ובחרנו לייצר דיאגרמת תוכן נפעל בהתאם לכללים הבאים:
  - גבולות המערכת מיוצגים ע"י היקף העיגול בו מופיעה שמה של המערכת.

---

[5] דיאגרמת תוכן

- מלבנים מחוץ לעיגול המערכת מייצגים יישות חיצונית למערכת[6]. יישות חיצונית יכולה להיות מערכת אחרת , ארגון אחר , שחקן , התקן חומרה וכד' – בהתאם לרשימת הישויות שסווגו כחיצוניות למערכת.
- הממשקים בין המערכת המפותחת לבין היישויות החיצוניות מיוצגים ע"י חצים עם תגיות הנקראים Flows. Flows מייצגים את תנועת הנתונים, אותות בקרה או אובייקטים פיזיים הנעים בין המערכת ליישויות החיצוניות.

דיאגרמת תוכן מספקת את הסקופ של הפרוייקט ברמת הפשטה גבוהה ובכך היא דומה מאוד לדיאגרמת ה Use Case.

- במידה ובחרנו לייצר דיאגרמת Use Case נפעל בהתאם לכללים הבאים:
  - היקף המלבן בדיאגרמת Use Case מייצג את גבולות המערכת והינו אנאלוגי להיקף העיגול בדיאגרמת התוכן.
  - ציורי האנשים מחוץ למלבן מייצגים שחקנים (יישויות חיצוניות) אשר הופיעו כמלבנים בדיאגרמת התוכן.
  - בשונה מדיאגרמת התוכן , דיאגרמת ה Use Case מספקת גם מידע (אמנם מוגבל) על פנים המערכת. כל אליפסה בתוך המלבן מייצגת Use Case.

מכיוון שבשלב הראשון של המתודולוגיה אנו עוסקים בתיחום של גבולות המערכת ניתן להסתפק בדיאגרמת תוכן עם יישויות חיצוניות כלליות ללא מידע על פנים המערכת. בשלבים מתקדמים יותר ניתן יהיה כבר לזהות שחקנים ו Use Cases ולייצר Use Case Diagram על בסיס דיאגרמת התוכן ומידע נוסף שהצטבר.

3.4.3. <u>קלטים</u>:

- ❖ תוצר הצהרת הבעיה.
- ❖ תוצר רשימת ישויות.
- ❖ תוצר סקופ פונקציונלי.

4.4.3. <u>כלים וטכניקות</u>:

- ❖ שימוש בכלי גרפי התומך ביצירת דיאגרמות Context / Use Case (לדוגמא: Rational Rose , Microsoft Visio)

5.4.3. <u>איכות</u>:

---

[6] נקראים גם טרמינטורים

08/01/09

❖ קווים מנחים לבדיקת איכות של <u>דיאגרמת תוכן</u> :

- הדיאגרמה תורכב מ:
  - יישויות חיצוניות (טרמינטורים המסומנים כמלבנים)
  - אפיקי מידע (Data Flows)
  - תהליך יחיד המסומן באמצעות עיגול ובתוכו שם המערכת המפותחת
- יש לוודא כי אין חיצי מידע בין ישויות חיצוניות לבין מאגרי מידע.
- יש לוודא כי אין חיצי מידע בין מאגר מידע אחד למשנהו.
- הישויות מחוברות למערכת באמצעות חיצי מידע המכילים את תיאור המידע המועבר והכיוון.
- יש לוודא עקיבות בין הופעת האלמנטים בתרשים להופעתם בתוצרים הקודמים (הצהרת הבעיה,רשימת יישויות,סקופ פונקציונלי).

❖ קווים מנחים לבדיקת איכות של <u>דיאגרמת Use Case</u>

(Rosenberg and Scott, 2001) :

- הדיאגרמה תורכב מ:
  - מלבן המסמן את גבולות המערכת ומכיל את שם המערכת.
  - ישויות חיצוניות (מיוצג ע"י ציור של "איש-מקל" ומתחתיו רישום של שם היישות)
  - Use Cases המסומנים כאליפסות ובתוכן שם ה Use Case.
  - קווים מקשרים (בין הישויות לבין ה Use Cases)
- שמות ה Use Cases יתחילו בשם פועל. כאשר השם יילקח מהטרמינולוגיה של התחום בו עוסקת המערכת (Domain Terminology).
- הUse Cases העיקריים ימוקמו בצד השמאלי העליון של הדיאגרמה.
- מיון משני של ה Use Cases יתבצע ע"פ סדר תזמון הגיוני של הפעולות , כאשר הUse Case שצפוי להתרחש ראשון יופיע מעל זה שצפוי להתרחש בהמשך.
- שחקנים ראשיים יופיעו בצד שמאל העליון של הדיאגרמה.
- שחקנים יופיעו מחוץ למלבן המערכת.
- שמות השחקנים יורכבו משם יחיד הקשור לתחום המערכת.
- כל שחקן מקושר באמצעות קו מחבר ללא חץ לUse Case אחד או יותר.
- אין קישורים בין שחקנים (אך ורק יחס הכללה אפשרי)
- יחסים אפשריים בין Use Cases הינם הכלה, הרחבה או הכללה והם מסומנים עם <<sterotype>> בהתאמה .

08/01/09

♦ <u>אימות התרשים</u>:

√ יש לוודא כי תרשים ה UML הינו תקני ובהתאם לסינטקס של UML.

√ יש לוודא כי בכל תרשים יש לפחות שחקן אחד המקושר לבועת Use Case אחת לפחות.

√ יש לוודא כי השחקן אכן חיצוני למערכת ואיננה חלק ממנה (כגון בסיס-נתונים פנימי).

√ אלמנטים הקרובים זה לזה מבחינה סמנטית יש לצייר קרובים זה לזה פיזית.

√ יש לצמצם למינימום את מספר הקווים אשר חותכים זה את זה.

√ יש לתת שם לדיאגרמה בהתאם לשם ה Use Case .

√ יש לבדוק האם יש צורך להוסיף בדיאגרמה שימוש ב Use Case אחר (קשר include).

√ יש לבדוק האם ה Use Cases המתואר בדיאגרמה הוא למעשה הרחבה של Use Case אחר (במקרה זה , יש להוסיף קשר extends).

√ יש לוודא עקיבות בין הופעת האלמנטים בתרשים להופעתם בתוצרים הקודמים (הצהרת הבעיה,רשימת יישויות,סקופ פונקציונלי).

♦ <u>טיפים ליצירת דיאגרמת Use Case</u>:

! על הדיאגרמה להתמקד באספקט מסויים של אינטרקציה מול המערכת.

! הדיאגרמה תכלול רק את ה Use Cases והשחקנים אשר חיוניים לצורך הבנת האספקט הנ״ל.

! הדיאגרמה צריכה לספק פרטים בצורה עקבית עם רמת ההפשטה הנדרשת. יש לחשוף רק התנהגויות אשר חיוניות לצורך הבנת האספקט בו מתמקדים.

! הדיאגרמה צריכה להכיל גם פרטים סמנטיים חשובים אשר יסייעו לקורא להבין את הקונטקסט.

6.4.3. סיכונים ושגיאות נפוצות:

❖ שגיאות נפוצות ביצירה של Use Case Diagrams:

השגיאה	טכניקת מניעה
התרשים כולל מרכיבים שאינם חלק מנוטציות UML.	יש לוודא כי תרשים ה UML הינו תקני ובהתאם לסינטקס של UML. רצוי להעזר בכלי התומך באופן מלא בסטנדרט UML.
ישנם שחקנים שאינם מקושרים לאף Use Case ואינם הכללה של שחקן אחר.	יש לוודא כי בתרשים יש לפחות שחקן אחד המקושר לבועת Use Case אחת לפחות.
שחקן המופיע מחוץ לגבולות המערכת הינו למעשה חלק מהמערכת המפותחת.	יש לבצע בדיקה עבור כל שחקן כי הוא אכן חיצוני למערכת ואיננו חלק ממנה (כגון בסיס-נתונים פנימי). פעולה זו כדאי לבצע שוב לאחר סיום השלב הבא במתודולוגיה.
ישנם אלמנטים "קרובים" המופיעים רחוק זה מזה בתרשים.	אלמנטים הקרובים זה לזה מבחינה סמנטית יש לצייר קרובים זה לזה פיזית.
ישנם קווים מצטלבים רבים בתרשים.	יש לצמצם למינימום את מספר הקווים אשר חותכים זה את זה.
לדיאגרמת ה Use Case אין שם.	יש לתת שם לדיאגרמה בהתאם לשם ה Use Case העיקרי אותו היא מתארת.
ישנו Use Case נוסף המופעל באחד מצעדי הUse Cases המתוארים אך הוא איננו מופיע בתרשים.	יש לבדוק האם יש צורך להוסיף בדיאגרמה שימוש ב Use Case אחר (קשר include).
ישנו Use Case המתואר בתרשים כעצמאי ומחובר ישירות לשחקן אך הוא למעשה הרחבה של Use Case אחר.	יש לבדוק בעבור כל Use Case בדיאגרמה האם הוא למעשה הרחבה של Use Case אחר (במקרה זה , יש להוסיף קשר extends).

טבלה5

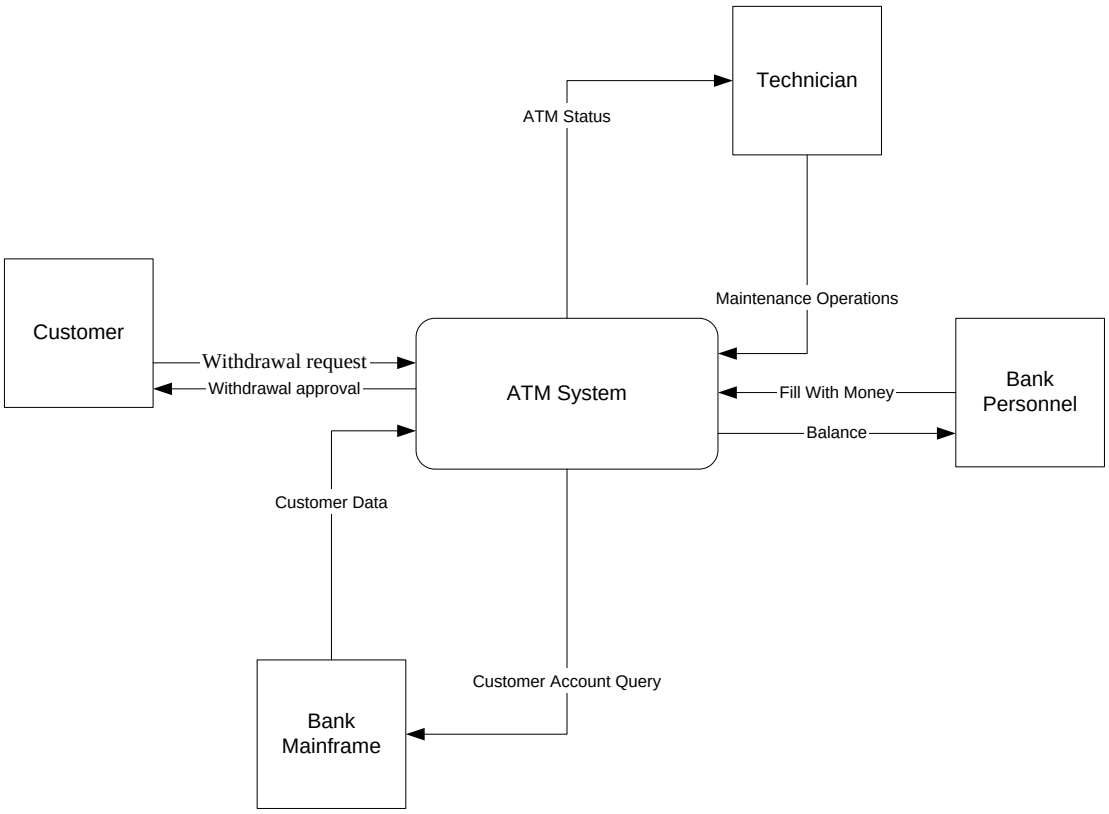
❖ <u>שגיאות נפוצות בנושאי סקופ וגבולות המערכת</u> (Kulak & Guiney, 2000) :

השגיאה	מה עשוי להגרם / הנחיות
יצירת Use Cases מבפנים החוצה.	Use Cases אלו נכתבים מהפרספקטיבה של האפליקציה ולא של השחקן. המשתמש/שחקן בד"כ איננו מבין את "הקרביים" של המערכת ופרספקטיבה זו איננה טבעית עבורו.
הכנסת פרטים של ממשק משתמש ב Use Cases.	בזמן איסוף הדרישות יש להשאיר פרטי ממשק משתמש מחוץ ל"משחק". פרטים אלו ניתנים להוספה רק לאחר סיום קביעת גבולות המערכת ואיסוף הדרישות העיקריות.
הרחבת גבולות המערכת יתר על המידה.	יש להכניס רכיבים לתוך המערכת רק אם תכולתם הינה חלק מהמערכת המפותחת. אם זו מערכת נפרדת המפותחת ע"י גוף אחר יש לסמנה כשחקן.
יצירת אינטרקציות מיותרות בין שחקן לבין Use Cases.	באופן כללי ה Use Case צריך לספק מטרת שחקן ובכך הינו בעל ערך עבור השחקן. השחקן לא בהכרח זה שמספק את הקלט ל Use Case. אם זה לא המצב אין טעם ליצור קשר בין שחקן ל Use Case.

טבלה6

7.4.3. <u>דוגמא לתוצר דיאגרמה מערכתית</u> :

להלן דוגמא **לדיאגרמת תוכן** עבור מערכת כספומט בבנק [7] :

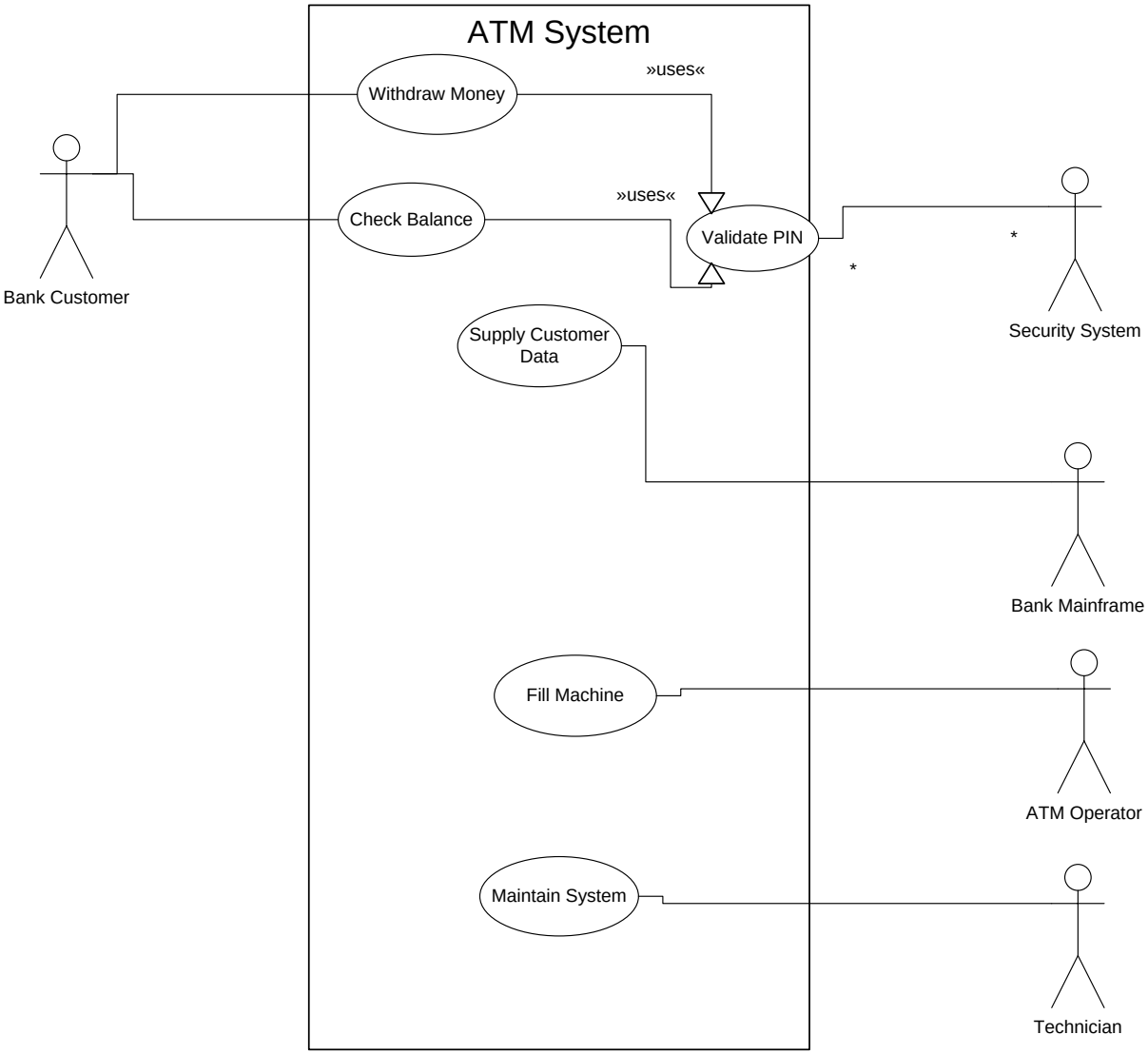


איור5

[7] ATM (Automatic Teller Machine)

08/01/09

להלן דוגמא **לדיאגרמת Use Case** עבור מערכת כספומט בבנק :



איור6

ניתן לראות כי הדיאגרמות אנלוגיות זו לזו :

המערכת (ATM) מיוצגת ב-2 הדיאגרמות במלבן הראשי. היישויות החיצוניות למערכת אשר יוצרות עימה אינטרקציה (לקוח,טכנאי,מפעיל ומחשב ראשי) מיוצגות בדיאגרמת התוכן בתור מרובעים בעוד שבדיאגרמת Use Case הן מיוצגות בתור ציור של "אנשי מקל".

מהנתונים הללו ניתן לגזור את גבולות המערכת והממשקים.

ישנם נתונים נוספים אשר לא זהים בין הדיאגרמות :

בדיאגרמת התוכן ניתן לראות מהם הנתונים הזורמים על אפיקי המידע השונים בין המערכת ליישויות החיצוניות ובכך לקבל את קונטקסט המערכת ביחס לעולם החיצוני.

בדיאגרמת ה Use Case ניתן לראות את ה Use Cases העיקריים המשורטטים כאליפסות ומכך ניתן ללמוד על התנהגות המערכת.

ב-2 המקרים ניתן להסיק מהי הפונקציונאליות העיקרית הנדרשת מהמערכת.

סיכום:

שלב תיחום גבולות המערכת מייצר מסגרת לשלבים הבאים. בשלב זה זיהוי מוחלט של היישויות וה Use Cases איננו במוקד אלא חשוב יותר לזהות מה כלול במערכת ומה חיצוני למערכת. תהליך זה מקנה הבנה של הבעיה שהמערכת באה לפתור. בתום שלב זה גבולות המערכת והסקופ הפונקציונלי ברורים. ישנה רשימה כללית של ישויות במערכת ומחוצה לה. מיקום המערכת ביחס ל"שאר העולם" ברור ועל כן השלב הטבעי הבא הינו זיהוי של הישויות אשר המערכת צפויה לתקשר עמם.

❖ בעלי תפקידים:

להלן רשימה של בעלי התפקידים המעורבים בשלב הנוכחי:

- מנתח דרישות – מייצר את רשימת היישויות אשר זוהו וכן את דיאגרמת ה Context / Use Case.
- מנהל הפרוייקט / מנהל מוצר – מספק ומעדכן את "הצהרת הבעיה".
- הנהלה בכירה – אימות "הצהרת הבעיה" , יצירת מסמך חזון המערכת.
- מנהל תצורה – דואג להכנסת התוצרים תחת ניהול תצורה ומתן תגיות סימון.
- איש הבטחת איכות – מוודא כי התוצרים תקינים בהתאם להנחיות האיכות.

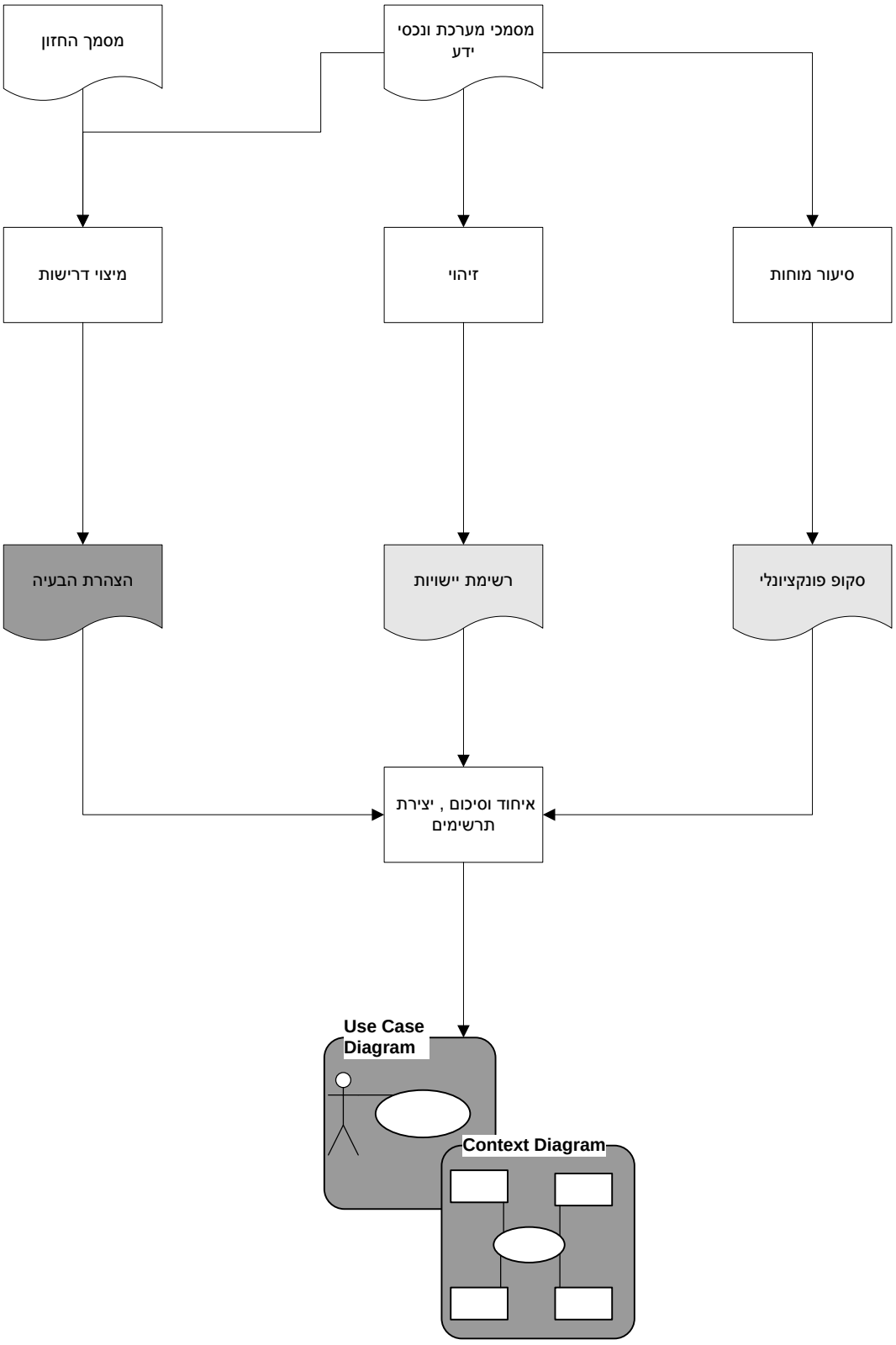
❖ רשימת תוצרים:

להלן רשימה של התוצרים השונים המופקים בשלב זה:

התוצר	**שלם/חלקי**	**צפי להשלמת התוצר**
הצהרת הבעיה	שלם	לא נדרש
רשימת יישויות	חלקי	בשלב הבא (זיהוי שחקנים ומטרות)
סקופ פונקציונלי	חלקי	בשלב הבא (זיהוי שחקנים ומטרות)
דיאגרמת תוכן/UC מערכתית	שלם	לא נדרש

טבלה7

❖ להלן תרשים מסכם לשלב I - תיחום גבולות המערכת :



4. שלב II - זיהוי שחקנים ומטרות:

<u>רציונל</u>:

כאשר אנו מגיעים לשלב זה הבעיה אותה אנו ממדלים כבר ברורה יותר, ישנה רשימה של ישויות ודיאגרמת Use Case מערכתית (או דיאגרמת תוכן) המתארת את גבולות המערכת.

בשלב זה נרצה לזהות את השחקנים במערכת ולהבין מהי מטרתו של כל שחקן. מודל ה Use Case הינו מודל אשר מונע מטרות (Goals Driven) , לאחר שנבין ונאפיין את כל מטרות השחקנים מהמערכת נוכל לאפיין את דרישות המערכת.

בתום זיהוי השחקנים, מטרותיהם ומתן עדיפויות יתקבלו דרישות פונקציונאליו ת ברמת דיוק ראשונה (Cockburn, 2000).

1.4. תוצר נדרש: רשימת פרופיל שחקנים (תוצר שלם)

רשימת פרופיל שחקנים זו רשימה המכילה את שמות כל השחקנים שזוהו כולל הסיווג של כל אחד מהם ותיאור הרקע והכישורים שלו הרלוונטיים למערכת.

1.1.4. <u>המטרה</u>:

אפיון וסיווג השחקנים ומטרותיהם.

2.1.4. <u>פעילויות</u>:

❖ זיהוי שחקנים

נבצע מעבר על רשימת הישויות משלב קודם ונאפיין עבור כל יישות מיהו השחקן אשר מייצג אותה. שחקנים הינם כל דבר אשר מתממשק עם המערכת שלנו כאשר כל שחקן מייצג תפקיד מסוים. כל יישות אשר זוהתה בשלב "תיחום גבולות המערכת" כיישות חיצונית למערכת עשויה להיות מיוצגת ע"י שחקן אחד או יותר וזאת בהתאם לתפקידה ומטרותיה. לדוגמא: אדם מסויים יכול להיות פעם אחת לקוח ובפעם אחרת עובד ולכן ייוצג ע"י שני שחקנים שונים בהתאם לתפקידו בכל פעם ובהתאם למטרותיו השונות. בהמשך נוסיף עמודה לטבלת השחקנים בשם "המטרה" וננסה להבין מהי המטרה/האינטרס של כל שחקן מהמערכת.

❖ זיהוי שחקנים נוספים

בעזרת פעילות זו ננסה לזהות שחקנים נוספים אשר לא זוהו כחלק מאותן יישויות אשר נסרקו בפעילות הקודמת.

לצורך כך ניתן להיעזר ברשימת השאלות הבאות (Armour and Miller, 2000):

- ? מי משתמש במערכת?
- ? מי מתקין את המערכת?
- ? מי מתניע את המערכת?
- ? מי מתחזק את המערכת?
- ? מי סוגר את המערכת?
- ? אילו מערכות אחרות משתמשות במערכת?
- ? מי מקבל אינפורמציה מהמערכת?
- ? מי מספק אינפורמציה למערכת?
- ? האם ישנם דברים שקורים אוטומטית במערכת? האם יש צורך בשחקן שהינו "זמן"?

❖ אפיון השחקנים

בפעילות זו ניצור פרופיל לכל שחקן. פרופיל השחקן מכיל את הרקע שלו והכישורים שלו בהקשר לשימוש שלו במערכת.

❖ סיווג השחקנים

בפעילות זו נסווג את השחקנים השונים שזוהו למספר קבוצות:

- *בעלי עניין*. בעל ענין זה מישהו או משהו שיש לו אינטרס מההתנהגות של ה-Use Case. בעל העניין משתתף בחוזה.

בדוגמא של מערכת ה ATM – המפקח על הבנקים עשוי להיות בעל עניין למרות שאיננו שחקן ואיננו בא במגע עם המערכת.

- *שחקן ראשי*. השחקן הראשי של ה Use Case הוא בעל העניין אשר מפעיל את המערכת על מנת לקבל שירות מסויים. ע"י פעולה זו השחקן הראשי מנסה לספק את מטרתו.

בדוגמא של מערכת ה ATM – הלקוח הינו השחקן הראשי ומטרתו למשוך מזומנים מחשבונו.

- *שחקן משני*. שחקן משני נקרא גם *שחקן תומך*. זהו שחקן חיצוני אשר מספק שירות למערכת. לדוגמא: מדפסת מהירה, Web-Service , קבוצת חוקרים שצריכה להחזיר מידע אשר תורם למערכת וכד'. אותה יישות יכולה להופיע כשחקן ראשי ב-Use Case אחד וכשחקן משני ב-Use Case אחר.

בדוגמא של מערכת ה ATM – המחשב הראשי של הבנק אשר מספק מידע על חשבונות הלקוחות הינו שחקן תומך.

3.1.4. <u>קלטים</u>:

❖ הצהרת הבעיה – תוצר משלב קודם, לאורו אנו פועלים לאורך כל הדרך. מסייע בהבנת האינטרסים של שחקני במערכת.

❖ תרשים Context/Use Case – תוצר משלב קודם המתווה עבורנו את גבולות המערכת. בעזרתו ניתן להבין האם הישות הינה חלק מהמערכת או שהינה שחקן המתקשר עם המערכת.

❖ רשימת יישויות חלקית – תוצר משלב קודם עליו נתבסס כנקודת יציאה לזיהוי ואיתור שחקנים.

4.1.4. כלים וטכניקות:

- ❖ שימוש ברשימת שאלות מנחות (עבור זיהוי שחקנים נוספים)
- ❖ ראיונות עם בעלי עניין

5.1.4. איכות:

- ❖ להלן קווים מנחים לבדיקת איכות התוצר:

- השחקן אכן חיצוני למערכת ואיננו חלק מהמערכת.
- במידה וקיימת מערכת נוכחית: המערכות המתממשקות אליה כיום ואשר רלוונטיות גם למערכת החדשה זוהו כשחקנים.
- פרופיל השחקן מתאר את הרקע שלו ואת כישוריו. במידה והשחקן הינו מערכת הפרופיל מתאר את היכולות העיקריות של המערכת.
- פרופיל השחקן מתואר באופן קצר ותמציתי.
- שם השחקן מיצג את תפקידו ואחריותו ביחס למערכת.
- כל השחקנים שונים זה מזה בתפקידם ביחס למערכת. אין 2 שחקנים אשר תפקידם כמעט וזהה.
- שם השחקן איננו כללי מדי ואיננו מכיל בתוכו שחקן נוסף.

6.1.4. סיכונים ושגיאות נפוצות:

השגיאה	טכניקת מניעה
שחקן המופיע בטבלת השחקנים הינו למעשה חלק מהמערכת המפותחת.	יש לבצע בדיקה עבור כל שחקן כי הוא אכן חיצוני למערכת ואיננו חלק ממנה (כגון בסיס-נתונים פנימי).
ישנה מערכת אשר אמורה לספק/לקבל מידע מהמערכת המפותחת אך איננה נכללת ברשימת השחקנים	יש לבצע מעבר גם על שחקנים אנושיים וגם על מערכות חיצוניות כולל מערכות Legacy אשר צפויות לקבל ו/או לספק מידע מהמערכת המפותחת.
שחקן תומך מזוהה בטעות כשחקן ראשי.	יש לוודא כי הפעילות אשר מבצע השחקן באה לשרת את מטרתו ולא מטרה של שחקן אחר. לדוגמא: פקיד או טלפנית המתניעים את התהליך בעבור שחקן ראשי.
שם השחקן מייצג את תפקידו בארגון אך לא ביחס למערכת המפותחת. (לדוגמא: מפעיל ATM עשוי להיות אדמיניסטרטור מחשבים בסניף)	יש לוודא כי שם השחקן מייצג את תפקידו ואחריותו כלפיי המערכת המפותחת.
שם השחקן נגזר משם ה Use Case אך איננו מספיק כללי. (לדוגמא: "מושך מזומנים" במקום "לקוח ATM")	יש לנסות למצוא את השם המתאים ביותר שאיננו ספציפי מדיי ועדיין משקף את תפקידו של השחקן ביחס למערכת.

טבלה8

7.1.4. דוגמא לתוצר רשימת פרופיל-שחקנים :

שם השחקן	סיווג	פרופיל (רקע וכישורים)
לקוח	שחקן ראשי	אדם מן היישוב אשר מסוגל להשתמש במסך-מגע. עשוי להיות בעל קושי לקרוא, קצר ראייה, עיוור צבעים וכד׳. לאדם זה קיים חשבון באחד מן הבנקים המקושרים למערכת.
מפעיל ATM	שחקן משני	עובד אשר באחריותו לדאוג כי הכספומט יהיה מלא במזומנים ובדפי מדפסת. באחריותו לפקח על הפעילות השוטפת והתקינה של המכונה.
טכנאי ATM	שחקן משני	אדם בעל כישורים טכניים ובעל ידע רחב ומקיף על אופן פעולת מכונת ה ATM. מסוגל לאתר ולתקן תקלות , כולל החלפת חלפים במידת הצורך.

טבלה9

2.4. תוצר נדרש: רשימת שחקן-מטרה (תוצר שלם)

רשימת שחקן-מטרה זו רשימה המורכבת משמות כל השחקנים שזוהו וכוללת את מטרתו של כל שחקן והעדיפות ביחס לשאר המטרות.

1.2.4. המטרה:

זיהוי מטרות השחקנים השונים.

2.2.4. פעילויות:

- ❖ זיהוי מטרות השחקנים הראשיים

בעזרת פעילות זו נזהה מהי מטרתו של כל שחקן ראשי. עבור כל שחקן ראשי נשאל את השאלות הבאות:

- ? מה השחקן רוצה להשיג מהמערכת?
- ? מהו האינטרס שמניע את השחקן?
- ? מדוע השחקן יוזם פעולה במערכת?
- ? מדוע השחקן מקבל מידע מן המערכת?
- ? מדוע השחקן מספק מידע למערכת?

רצוי לבצע ראיונות עם בעלי העניין על מנת לנסות להבין את מטרות השחקנים ולנסות לזהות שחקנים נוספים בהתאם למידע המסופק מבעלי העניין בראיון.

- ❖ מתן עדיפויות למטרות השחקנים

בפעילות זו נתעדף את מטרות השחקנים. העדיפות מסייעת להבין מהן המטרות העיקריות והחשובות ואילו מטרות הן מטרות משנה. בהמשך ניתן יהיה להיעז ר בעדיפויות השונות לצורך חלוקת ה Use Cases לאיטרציות ובחלוקתם לצוותי פיתוח שונים. רצוי לקבוע סולם עדיפויות בין 1 (הכי גבוה) ל 5 (הנמוך ביותר).

- ❖ יצירת רשימת שחקנים אחודה

פעילות זו הינה פעילות טכנית פשוטה ובה נאחד את התוצר הנוכחי עם התוצר הקודם ונקבל טבלה אחת המכילה את השחקנים , הפרופיל שלהם ומטרתם. בפעילות זו נשמיט את עמודות הסיווג והעדיפות.

3.2.4. קלטים:

- ❖ רשימת פרופיל שחקנים (תוצר קודם משלב זה).

4.2.4. כלים וטכניקות:

- ❖ שימוש ברשימת שאלות מנחות (עבור זיהוי שחקנים נוספים)
- ❖ ראיונות עם בעלי עניין

5.2.4. איכות:

- להלן קווים מנחים לבדיקת איכות התוצר:
  - מטרת השחקן איננה ברמה נמוכה מדיי אלא משקפת את האינטרס האמיתי אותו ברצונו להשיג מהמערכת.
  - מטרת השחקן מתוארת באופן קצר ותמציתי.
  - מטרה עיקרית של שחקן מסוים תהיה בעלת עדיפות גבוהה יותר ממטרות המשנה שלו.
  - מטרת שחקן ראשי / בעל עניין תהיה בעלת עדיפות גבוהה משל מטרת שחקן משני.

6.2.4. סיכונים ושגיאות נפוצות:

השגיאה	**טכניקת מניעה**
מטרת השחקן איננה משקפת את מטרתו אלא את מטרתו של שחקן אחר.	שימוש בשאלות המנחות ובדיקת חפיפה אל מול שחקנים בעלי מטרות דומות.
שם השחקן כללי מדיי ולמעשה טומן בחובו מספר שחקנים. (לדוגמא: "משתמש" כאשר ישנם מספר משתמשים עם מטרות שונות ביחס למערכת)	עבור כל שחקן יש לוודא כי איננו מכיל מטרות רבות אשר חלקן שייכות למעשה לשחקנים אחרים.
מטרת משנה מופיעה עם עדיפות גבוהה משל המטרה העיקרית של השחקן.	בהנתן מספר מטרות לאותו שחקן יש לבחון מהי המטרה הראשית שאם נוותר עליה כל מהות השחקן תשתנה.

טבלה10

7.2.4. <u>דוגמא לתוצר רשימת שחקן-מטרה</u>:

השחקן	המטרה	עדיפות
מפעיל ATM	חידוש מלאי המזומנים במכונה.	2
	חידוש מלאי ניירת ושאר חומרי אחזקה במכונה	2
טכנאי ATM	הבאת מכונת ה ATM למצב תקין.	3
	הרצת בדיקות עצמיות למכונה.	4
לקוח	שימוש במכונת ה ATM	1
	משיכת מזומנים	1
	הפקדת מזומנים	2
	העברת כספים לחשבון אחר	2
	בירור יתרה	2

טבלה11

8.2.4. <u>דוגמא לתוצר רשימת שחקנים-אחודה</u>:

שם השחקן	פרופיל	מטרה
לקוח	אדם מן היישוב אשר מסוגל להשתמש במסך-מגע. עשוי להיות בעל קושי לקרוא, קצר ראייה, עיוור צבעים וכד׳. לאדם זה קיים חשבון באחד מן הבנקים המקושרים למערכת.	משיכת מזומנים באופן מהיר ופשוט מתוך חשבונו בבנק. מטרות נוספות: בירור יתרת חשבון , הפקדת צ׳קים , העברת כספים מחשבון לחשבון.

טבלה12

סיכום:

התוצרים של שלב זיהוי השחקנים והמטרות בשילוב עם תוצרי שלב תיחום גבולות המערכת מהווים בסיס ליצירת Use Cases שלדיים. השחקנים חשובים בשלבים הראשוניים של איסוף הדרישות ובשלב הסופי לקראת מסירת המערכת. יצירת רשימה של שחקנים עוזרת לנו להתמקד במערכת כולה. מודל ה Use Case הינו מודל מונע-מטרות ולכן הדבר שמעניין אותנו למעשה אלו מטרות השחקנים. נסיון לחפש באופן ישיר את המטרות כנראה היה מפספס לא מעט מהן ולכן מציאת המטרות דרך מידול השחקנים הינה דרך טובה יותר. יצירה של מספר עודף של שחקנים אינה יכולה להזיק מאוד מכיוון שלכל היותר נקבל את אותן מטרות פעמיים. כאשר נבצע מעבר על הרשימה לצורך מתן עדיפות נגלה את הכפילות ונסיר אותה. ישנה סבירות כי גם בתום שלב זה עדיין ישנן מספר מטרות שהמערכת צריכה לתמוך בהן אך הן לא זוהו עדיין. מטרות נוספות נוטות להתגלות בשלב של טיפול בכשלונות של Use Case (Cockburn, 2000). מטרות אלו יטופלו בהמשך אך בשלב זה עלינו לנסות ולחשוף את מרבית המטרות כפי שתואר לעיל.

❖ בעלי תפקידים:

להלן רשימה של בעלי התפקידים המעורבים בשלב הנוכחי:

- מנתח דרישות – ניתוח רשימת הישויות והפיכתה לטבלת שחקנים ומטרות.
- בעליי עניין – השתתפות בראיונות לצורך הבנת האינטרסים שלהם.
- איש הבטחת איכות – מוודא כי התוצרים תקינים בהתאם להנחיות האיכות.

❖ רשימת תוצרים:

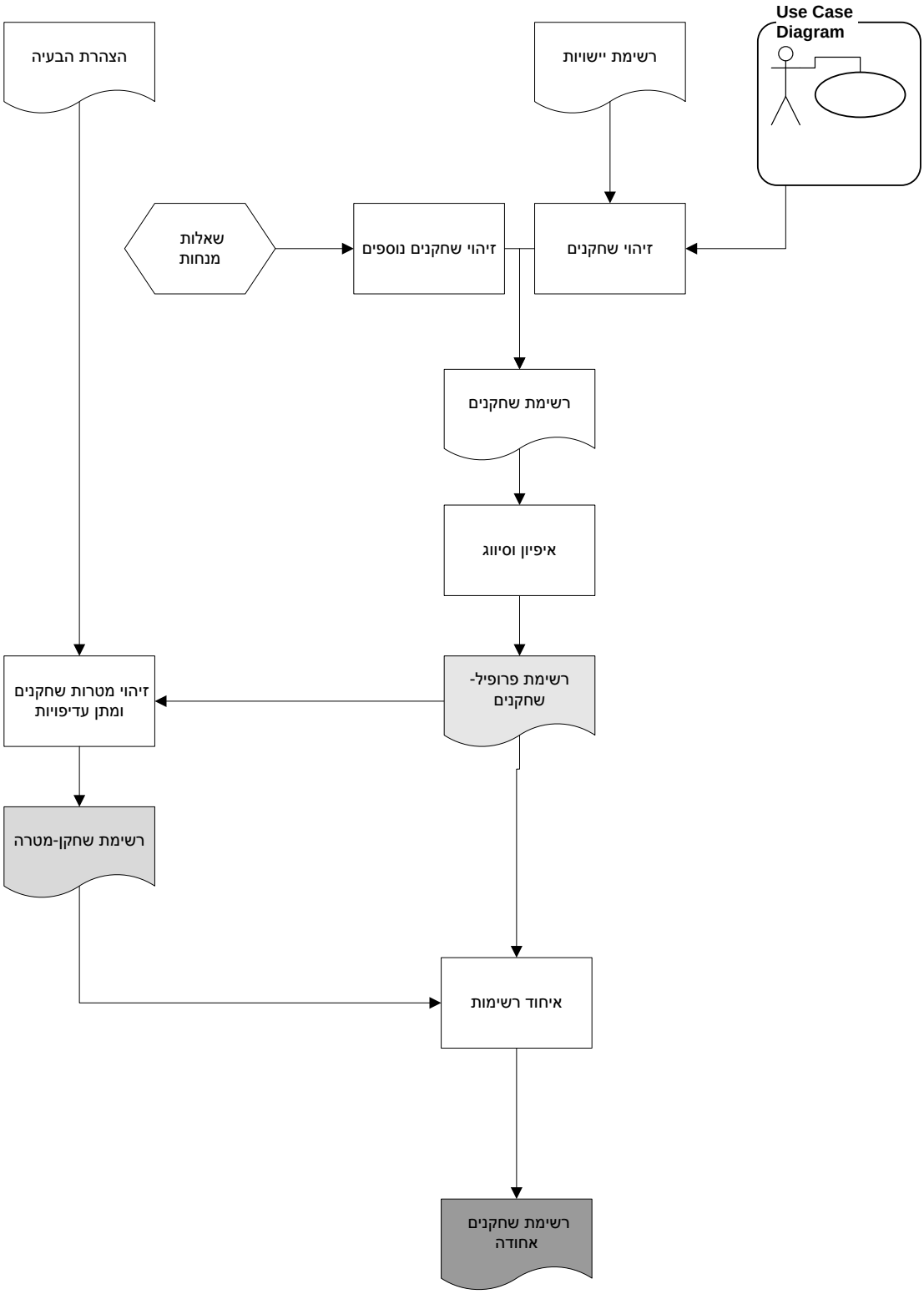
להלן רשימה של התוצרים השונים המופקים בשלב זה:

התוצר	שלם/חלקי	צפי להשלמת התוצר
רשימת פרופיל שחקנים	שלם	לא נדרש
רשימת שחקן-מטרה	שלם	לא נדרש

טבלה13

08/01/09

להלן תרשים מסכם לשלב II - זיהוי שחקנים ומטרות :



איור8

5. שלב III - גיבוש Use Case שלדי

<u>רציונל</u>:

מטרת שלב זה הינה ליצור מספר Use Cases עיקריים במערכת ללא פירוט רב. בהתאם לרעיונות המתודולוגיה אנו ממקדים את מרבית האנרגיה ב-Use Cases העיקריים של המערכת מבלי לרדת לפרטים הקטנים ביותר. Use Case שלדי הינו תמציתי ומכיל את עיקרי הדברים. בתום שלב זה יהיו בידינו שלדים של Use Cases אשר נחלקם לחבילות ובשלבים הבאים נעבה אותם. שלב זה יתבצע במספר איטרציות על מנת ליצור Use Cases שלדיים בכל הרמות.

בתום השלב יהיו בידינו תוצרים המהווים דרישות פונקציונאליות של המערכת ברמת דיוק שניה (Cockburn, 2000).

מכיוון שUse Case שלדי איננו עמוס בפרטים והינו תמציתי יהיה זה אך טבעי להציגו בפני בעלי העניין השונים ובכך לסקור את דרישות המערכת מבלי להלאות בפרטים מיותרים. חשוב לזכור כי Use Case ובפרט Use Case שלדי צריך להיות ברור לקריאה עבור בעלי עניין המגיעים מתחומים שונים ואיננו צריך להכיל מושגים טכניים מעולם המחשוב כפי שקורה לעיתים תכופות ע"י כותבים לא מנוסים המגיעים מעולם מדעי המחשב. (Jagielska D., Wernick P., Wood M. and Bennet S., 2006)

1.5. תוצר נדרש: Use Cases שלדיים (תוצר חלקי)

Use Case שלדי הינו Use Case חלקי המכיל את המהות העיקרית של ה Use Case. Use Case שלדי מכיל תיאור כללי של תרחיש ההצלחה הראשי.

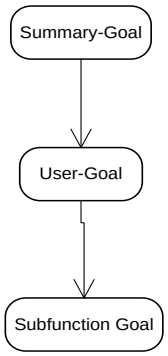
1.1.5. המטרה:

יצירת Use Cases שלדיים בהתאם לרמה המטופלת באיטרציה הנוכחית.

המתודולוגיה מציעה לבצע לפחות 3 איטרציות כאשר בכל איטרציה אנו מטפלים ב Use Cases ברמה שונה (ראה 9.5.5):

- איטרציה ראשונה – יצירת Use Cases ברמת Summary-Goals[8].
- איטרציה שניה – יצירת Use Cases ברמת User-Goals[9].
- איטרציה שלישית – יצירת Use Cases ברמת Subfunctions-Goals[10].

החלוקה לרמות השונות מאפשרת לעבוד בצורה איטרטיבית Top-Down .



איור9

[8] רמת העל במודל ובה מספקים את האינטרסים של מספר שחקנים ראשיים במערכת. משך זמן ביצוע של Use Case ברמה זו הינו בדרך כלל בסדרי גודל של שעות , ימים , חודשים.

[9] הרמה המרכזית במודל ובה השחקן הראשי או שחקן אחר מנסה לגרום למערכת לבצע עבודה מסויימת. משך זמן ביצוע של Use Case ברמה זו הינו בדרך כלל בסדרי גודל של דקות.

[10] הרמה התחתונה במודל: ה Use Cases נדרשים עבור קריאות המודל או כי Use Cases אחרים משתמשים בהם.

2.1.5. <u>פעילויות</u>:

❖ זיהוי Use Cases ברמה המטופלת [11]

לצורך ביצוע פעילות זו אנו נעזר ברשימת השחקנים המאוחדת אשר יצרנו בשלב "זיהוי שחקנים ומטרות". רשימת השחקנים עם המטרות שלהם מהווה אינדקס ליצירת Use Cases. באיטרציה הראשונה נחפש את ה- Use Cases החיצוניים ביותר (outermost) אשר יספקו את האינטרסים של מספר שחקנים ראשיים במערכת. נבצע מעבר על רשימת בעלי העניין והאינטרסים שלהם , כולל אלו שאינם באים במגע ישיר עם המערכת. נחפש את מטרות העל ונזהה דרך אילו שחקנים המטרות מושגות. באיטרציה השניה נעסוק ב- Use Cases של הרמה המרכזית במודל. ברמה זו השחקן הראשי או שחקן אחר מנסה לגרום למערכת לבצע עבודה מסויימת. השגה של המטרה ברמה זו משיגה מספר מטרות של רמה נמוכה יותר (Subfunctions). רשימה של Use Cases ברמה הזו מהווים למעשה תמצית של פונקציונליות המערכת כולה. מרבית ה Use Cases ברמה זו יזוהו ע"י מעבר פשוט על מטרות השחקנים הראשיים מתוך טבלת השחקנים האחודה. באיטרציה השלישית נעסוק ב-Use Cases של הרמה התחתונה במודל. אלו Use Cases שנדרשים בעיקר לצורך הקריאות של המודל (Readability) או בגלל ש-Use Cases אחרים משתמשים בהם.

להלן דוגמאות לרמות השונות של Use Cases:

- רכישת מניות Online תוך שימוש ב"stock trap" – רמת Summary Goal.
- משיכת מזומנים מהכספומט – רמת User Goal.
- עדכון יתרת הלקוח לאחר הפקדה – רמת Subfunction Goal.

[11] בהתאם לאיטרציה הנוכחית

❖ בחירת שם ל Use Case

לכל Use Case אשר זיהינו ברמה המטופלת יש לבחור שם מתאים (כמתואר ב9.5.2).

להלן קווים מנחים לבחירת שם מתאים ל Use Case :

- השם צריך להיות מונחה-מטרה (Goal-driven) ועליו לתאר את מטרת השחקן אותה ברצונו להשיג (לדוגמא: משיכת מזומנים). לצורך כך כדאי לשאול את השאלות הבאות:
  - ? מדוע השחקן יוזם את האינטרקציה עם המערכת?
  - ? מהי המטרה שברצונו להשיג כאשר הוא יוזם את הפעולות?
  - ? איזה ערך מוסף מתקבל עבור כל שחקן?
- השם צריך להכתב בצורת פעולה ולא כתיאור פאסיבי. על השם להתחיל בפועל (לדוגמא: Withdraw Cash ולא Cash Withdrawal , בעברית ההבחנה פחות ברורה). לצורך כך יש לחשוב על רשימת To-Do (לדוגמא: רשימת פעולות שברצוננו לבצע בבואנו לכספומט – Withdraw Cash, Transfer Funds , Deposit Funds).
- השם צריך להיות מספיק ברור עבור הקורא על מנת שיוכל להבין באופן כללי במה ה-Use Case עוסק. רצוי לא להשתמש בז׳רגון מקצועי אלא לתת שם אשר יובן ע״י אנשים מתחומים שונים (לדוגמא: הלקוח לא בהכרח מכיר את הז׳רגון של אנשי התוכנה).
- השם צריך להיות תמציתי ולא ארוך מדיי, בד״כ 2-3 מילים יספיקו.

❖ הכנת תבנית Use Case ומילוי סעיפים עיקריים

- תחילה נבחר את סוג התבנית איתה נעבוד לאורך כל המודל. ניתן לבחור בתבנית חלקית או בתבנית מלאה אשר בשלב הנוכחי נמלא סעיפים עיקריים בה וסעיפים אחרים נשאיר ריקים. בשלבים הבאים של המתודולגיה נמלא את יתר הסעיפים אשר נותרו ריקים.
- מילוי סעיף השחקנים - יש למלא בתבנית את סעיף השחקן הראשי המשתתף בהתאם לתפקידו (ראה 9.5.3). שם השחקן צריך להתאים לשמו כפי שמופיע בטבלת השחקנים האחודה.
- מילוי סעיף ה Scope – יש לציין מהו ה Scope של ה Use Case המטופל. ה Scope בסעיף זה הינו מתייחס לתכן (ראה o).
- מילוי סעיף הרמה – יש לציין באיזו רמה נמצא ה Use Case (ראה 9.5.5).

להלן דוגמא לקביעת הרמה בהתאם למטרה:

השחקן	המטרה	הרמה
מפעיל ATM	חידוש מלאי המזומנים במכונה. חידוש מלאי ניירת ושאר חומרי אחזקה במכונה	User Goal User Goal
טכנאי ATM	הבאת מכונת ה ATM למצב תקין. הרצת בדיקות עצמיות למכונה.	Summary Goal User Goal
לקוח	שימוש במכונת ה ATM משיכת מזומנים הפקדת מזומנים העברת כספים לחשבון אחר בירור יתרה	Summary Goal User Goal User Goal User Goal User Goal

טבלה14

- תיאור בקצרה של תרחיש ההצלחה הראשי – יש לכתוב בקצרה (3-2 משפטים) את תהליך ההצלחה הראשי ומה מושג בסופו. בשלב זה לא נפרט את התרחיש הראשי לצעדים ולא נתייחס לתרחישים אלטרנטיביים וטיפול בכישלונות. ראה דוגמא בסעיף תוצרים.

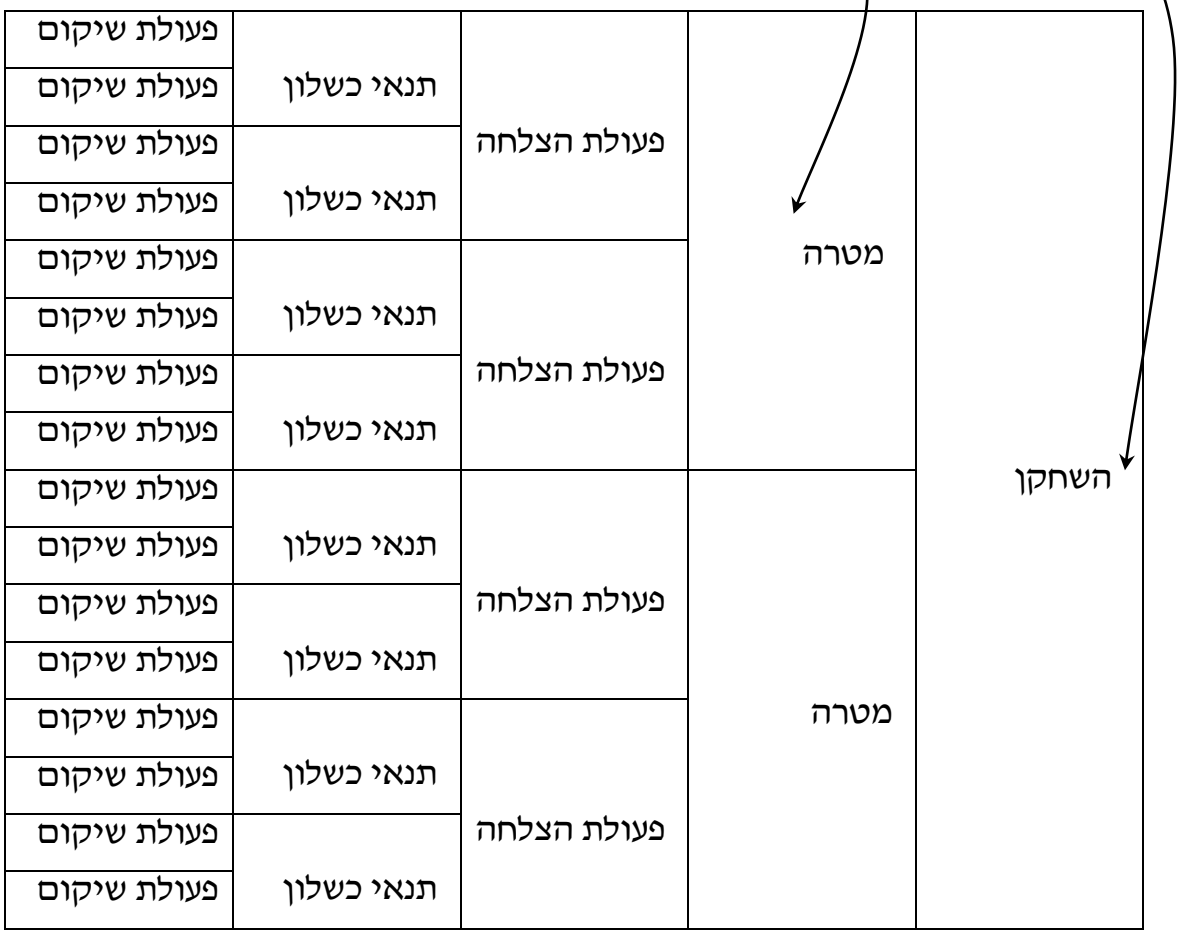
3.1.5. קלטים:

- ❖ רשימת שחקנים אחודה (שחקן-פרופיל-מטרה)
- ❖ תרשים Use Case ראשי

4.1.5. כלים וטכניקות:

- ❖ שימוש ברשימת השחקנים המאוחדת (תוצר משלב קודם) כאינדקס לחיפוש Use Cases כמתואר באיור הבא:

השחקן	פרופיל	מטרות



איור10

הערה: בשלב הנוכחי אנו מתארים בקצרה רק את פעולות ההצלחה בתרחיש ההצלחה הראשי.

❖ שימוש בתבניות Use Cases מוכנות מראש.

5.1.5. <u>איכות</u>:

❖ טבלה לבקרת איכות עבור סעיפי ה Use Case השלדי (Cockburn, 2000):

השדה בתבנית ה Use Case	שאלות הבקרה:
שם ה Use Case	1. האם השם הוא שם-פועל אשר מבטא את מטרתו של השחקן הראשי? 2. האם המערכת מסוגלת לספק את המטרה? 3. האם השם קצר תמציתי וברור?
שחקן ראשי	4. האם לשחקן הראשי יש התנהגות? 5. האם מטרת השחקן הראשי היא שירות המובטח ע"י המערכת? 6. האם ישנה עקיבות לטבלת השחקנים?
סקופ	7. האם ה Use Case מתיחס למערכת המוזכרת בסקופ כאל קופסה שחורה? 8. האם מבוצע תכן רק עבור תכולת המערכת המוזכרת בסקופ?
רמה	9. האם תוכן הUse Case אכן מתאים לרמה כפי שצויינה בסעיף זה? 01. האם המטרה באמת מתאימה לרמה שצויינה בסעיף זה?
תקציר תרחיש הצלחה ראשי	11. האם התרחיש מתואר החל מהטריגר ועד לסיום מוצלח? 21. האם סדר הפעולות המתואר הינו נכון? 31. האם התקציר מתמקד במטרה העיקרית ולא במטרות משנה?

טבלה15

❖ להלן מספר שאלות בקרה לתוצרי ה Use Cases השלדיים:

❖ האם לכל Use Case יש מספר מזהה ייחודי ?

❖ האם נעשה שימוש באותה תצורה של תבנית עבור כל ה Use Cases?

❖ האם ה Use Cases מנוהלים תצורה ? ישנם תגיות ? סומן "קו-בסיס" לכל איטרציה?

6.1.5. <u>סיכונים ושגיאות נפוצות</u>:

השגיאה	הסיכון	טכניקת מניעה
שם ה Use Case שגוי : איננו מתחיל בפועל (לדוגמא: Cash Withdrawal) , מעורפל וכללי מדיי (לדוגמא: Process / Do) , ארוך מדיי (לדוגמא: Deposit money into account using special envelope).	חוסר בהירות וקריאות של ה Use Case ע"י הלקוח / צוות הפיתוח ועקב כך שרשרת של תקלות בהמשך המידול. יצירת Use Case כללי מדיי / פרטני מדיי.	הצמדות לקווים המנחים למתן שם עבור Use Case.
ה Use Cases נכתבים מנקודת המבט של המערכת ולא של השחקנים. (לדוגמא: Cash Withdrawal במקום Withdraw Cash)	כתיבה פונקציונלית ועקב כך פספוס מטרות השחקן.	הצמדות למתודולוגיה אשר מכתיבה עבודה מתוך טבלת השחקנים כאינדקס.
שמות השחקנים לא עקביים (לדוגמא: שם השחקן בטבלת השחקנים איננו זהה לשמו כפי שמופיע בדיאגרמת החבילות)	חוסר בהירות בהבנת המודל.	שימוש בשמות השחקנים לאורך כל שלבי המודל כפי שנקבעו בטבלת השחקנים .
סקופ שגוי	יגרור לתיכון שגוי בהמשך.	בחינה של הסקופ בהתאם לשאלות הבקרה בסעיף האיכות.
רמה שגויה : רמה גבוהה / נמוכה מדיי עבור ה Use Case	עשוי לגרור פירוק יתר של Use Case ו/או פספוס של Use Cases אחרים.	בחינה של הרמה בהתאם לשאלות הבקרה בסעיף האיכות.
תקציר תרחיש ההצלחה ארוך מדיי / מתמקד במטרות משנה / סדר פעולות שגוי	אי בהירות בהבנת המודל ויצירת תגובת שרשרת של טעויות.	בחינה של תקציר התרחיש בהתאם לשאלות הבקרה בסעיף האיכות.

טבלה16

08/01/09

7.1.5. להלן דוגמא ל Use Case שלדי המתקבל בתום שלב זה :

Use Case #**3**	Use Case Name: **Withdraw Cash**	
Primary Actor	**Bank Customer**	
Scope	**ATM** (Automated Teller Machine)	
Level	**User-Goal**	
Brief Description (Can be deleted after "Description" is filled step by step)	**This use case describes how the Bank Customer uses the ATM to withdraw money from his/her bank account.**	
Stakeholders & Interests		
Precondition		
Minimal Guarantees		
Success Guarantees		
Trigger		
Description	Step	Action
	1	
	2	
	3	
Extensions	1a	
	1b	

טבלה17

2.5. תוצר נדרש: חבילות Use Cases (תוצר חלקי)

חבילות Use Case חלקיות הינן אוסף של Use Cases שלדיים בעלי מכנה משותף.

1.2.5. <u>המטרה</u>:

כאשר המערכת אותה אנו ממדלים (SuD) הינה מורכבת וגדולה, ייווצרו לנו כמויות Use Cases גדולות ולכן ישנו הכרח לבצע חלוקה לחבילות. החלוקה לחבילות מאפשרת בהמשך לחלק את העבודה לצוותי עבודה שונים. בשלב זה יש בידנו מספר Use Cases שלדיים בהתאם לרמה של האיטרציה המטופלת. Use Cases אלו יש לארגן ולנהל החל משלב יצירתם. פעולות שבירה/איחוד של Use Cases נבצע גם בשלב הבא כאשר הUse Cases מלאים וישנו מספיק "בשר" לקבלת ההחלטות.

2.2.5. <u>פעילויות</u>:

- **טיפול ב Use Cases מסוג CRUD**:

Use Cases מסוג CRUD הינם Use Cases אשר מבצעים פעולות יצירה, עדכון ומחיקה: "Create a new X", "Update an X", and "Delete an X". הכינוי CRUD נלקח מעולם בסיסי הנתונים הישן (Create,Replace,Update,Delete). בפעילות זו אנו נאתר Use Cases מהסוג הנ"ל ונאחדם לכדי Use Case יחיד בשם "Manage X" או שם דומה. האיחוד יתבצע באופן כזה שהתרחיש הראשי יכלול את Update וההסתעפויות יכילו את Create ו Delete. איחוד זה עוזר לצורך קריאות המודל ומקטין את הסיבוכיות.[12]

- **אריזת Use Cases לחבילות**:

בפעילות זו נבצע אריזה של מספר Use Cases בעלי מאפיינים משותפים לכדי חבילת Use Cases אחת. הפרמטרים אשר על פיהם ניתן לבצע את החלוקה הינם מגוונים: שייכות לשכבה בארכיטקטורה , שייכות לאיטרציה , שייכות לפונקציונליות וכו'. את החלוקה לחבילות נתאר בעזרת דיאגרמת חבילות תקנית בשפת UML. רצוי גם להוסיף טבלת חלוקה לחבילות בתצורה מילולית אשר תכיל את שמות החבילות , המאפיין של החבילה ואילו Use Cases כלולים בה.

---

[12] ישנן דעות הגורסות כי איחוד זה איננו טבעי ועשוי להוביל לבעיות במודל (ראה מאמר בשם How To Avoid Use-Cases Pitfalls שפורסם בשנת 2000 ע"י Susan Lilly באתר Dr.Dobb's Portal)

3.2.5. <u>קלטים</u>:

❖ Use Cases שלדיים (נוצרים בשלב זה בפעילות הקודמת).

4.2.5. <u>כלים וטכניקות</u>:

❖ שימוש בקווים מנחים לחבילות Use Cases (Bittner and Spence , 2002):

- שם החבילה: צריך להיות קצר ומתאר את המשותף ל Use Cases הארוזים בחבילה. בניגוד לשמות ה Use Cases , שם החבילה צריך להכתב בתצורה פאסיבית (לדוגמא: ATM Customer Services).
- בדיאגרמת החבילות נציין Stereotype מסוג <<use case package>> עבור החבילה.
- דיאגרמת חבילות ה Use Cases תכיל:
  - את כל הUse Cases המוכלים בחבילה.
  - את כל הקשרים בין ה Use Cases
  - את כל השחקנים המקושרים לUse Cases שבחבילות ואת הקשרים השונים.
- על דיאגרמת החבילות להכיל 7 ± 2 "בועות" (Class / Use Case).
- איגוד לחבילות יכול להתבצע במספר רמות (חבילות מקוננות) בהתאם למספר הרכיבים (שחקנים, Use Cases , חבילות אחרות):
  - 0-15: אין צורך באיגוד לחבילות.
  - 10-50: שימוש באיגוד לחבילות ברמה אחת.
  - יותר מ 25: שימוש באיגוד לחבילות בשתי רמות.
- על החבילות להיות בעלות לכידות (cohesive). הלכידות צריכה להיות בין ה Use Cases השונים באותה החבילה. בדיקת לכידות פשוטה הינה מציאת שם מתאר טוב וקצר לחבילה. אם קשה למצוא שם כזה אז ככל הנראה ישנה בעיית לכידות ויש לנסות לאתר את הUse Cases הלא מתאימים בחבילה.
- יש להימנע מתלות ציקלית בין החבילות השונות. התלות בין החבילות צריכה לשקף את היחסים הפנימיים.

5.2.5. <u>איכות</u>:

❖ שימוש בשאלות מנחות (התשובה צריכה להיות <u>כן</u> עבור כל השאלות):

? האם כל ה Use Cases מאורגנים בחבילות? האם ישנו תיעוד המציין את המפתח על פיו בוצעה החלוקה לחבילות?

? האם קיים תרשים חבילות?

? האם הייתה התייחסות ל Use Cases מסוג CRUD?

? האם ישנה לכידות בין ה Use Cases השונים באותה החבילה ?

? האם אין תלות מעגלית בין החבילות ?

? האם שם החבילה משקף את המשותף לUse Cases באותה החבילה ?

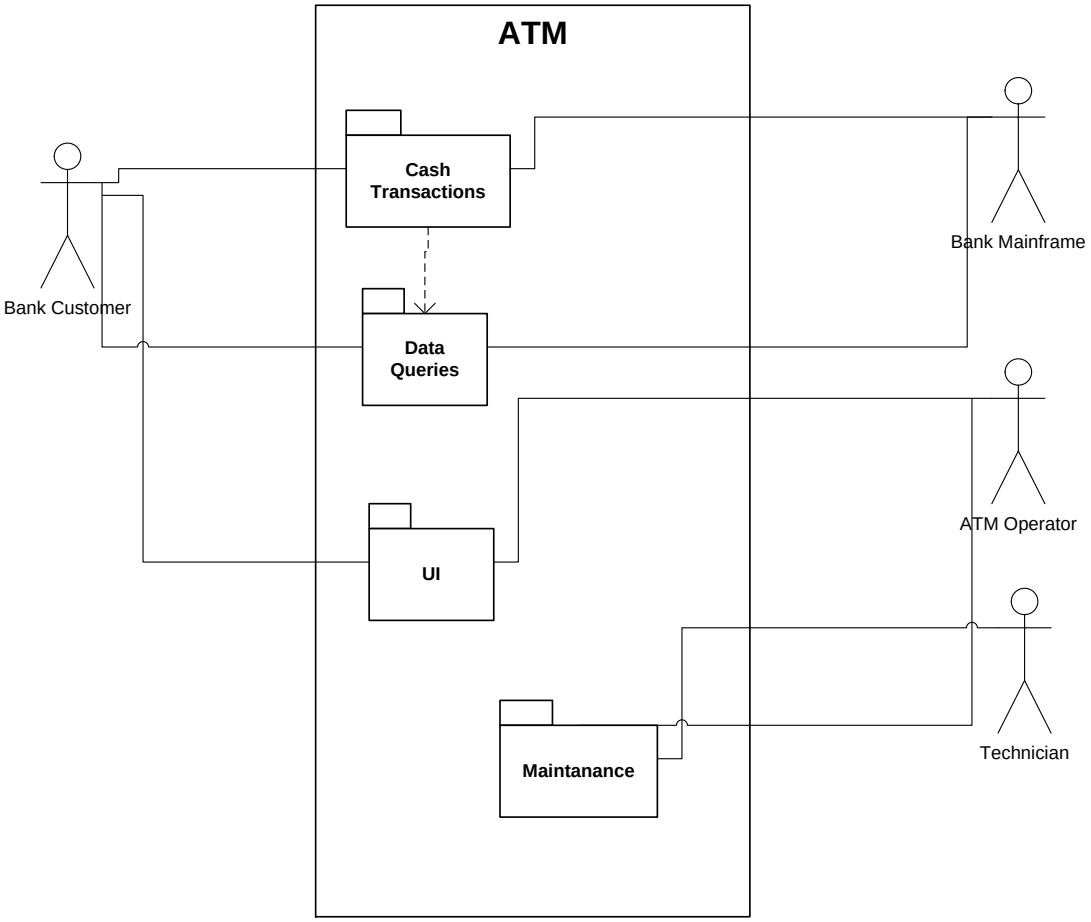
6.2.5. <u>סיכונים ושגיאות נפוצות</u> (Kulak and Guiney ,2000):

השגיאה	**מה עשוי להגרם / הנחיות**
תחזוקה של רשימת דרישות זמנית	לעיתים הצוות מתחזק רשימת דרישות זמנית ולאחר מכן מבצע מיפוי שלהם ל Use Case אשר יצר. כפילות זו גורמת לעבודה מיותרת ובה הצוות בעצם משנה את פורמט הדרישות מרשימה לפורמט של Use Cases. אין צורך להחזיק רשימות אלו.
אריזת Use Cases לחבילות בצורה לא טובה.	איגוד לחבילות חייב להתבצע ע״פ קריטריון משותף וכזה שבעלי העניין ימצאו בו הגיון. האיגוד יבוצע באמצעות דיאגרמת חבילות של UML ע״מ לאפשר הצגה ברורה.
אי שמירה של הUse Cases במסד נתונים.	שמירה במסד נתונים מאפשרת תחזוקה טובה של המודל. ללא מסד נתונים יהיה זה קשה מאוד לבדוק עקיבות וקשרים (cross-reference).
הכנסת הסתעפויות כחלק מתרחיש ההצלחה הראשי.	תרחיש ההצלחה הראשי צריך להיות תמציתי וברור. שגיאות והסתעפויות ייכתבו בהמשך בסעיף ההסתעפויות. ניפוח של תרחיש ההצלחה הראשי עם הסתעפויות יהפוך את ה Use Case לבלתי ברור ומסובך לקריאה.

טבלה18

08/01/09

להלן תרשים חבילות לדוגמא :



איור11

סיכום:

בשלב זה אנו מתחילים לראשונה ביצירת ה Use Cases עצמם. אנו יוצרים שלדי Use Cases אשר ימולאו בהמשך. שלב זה מבוצע בצורה איטרטיבית כאשר המפתח לכל איטרציה הינו הרמה של ה Use Case.

הפעילות מבוצעת בצורה שיטתית כאשר נעשה שימוש ברשימת השחקנים כאינדקס ממנו יוצאים. באמצעות חשיפת המטרות של כל שחקן זיהינו את הUse Cases וכך הצלחנו למלא אותם בצורה הדרגתית.

בשלב זה אנו מבצעים ארגון מבני של ה-Use Cases ומחלקים לחבילות.

❖ בעלי תפקידים:

- מנתח דרישות – מייצר את ה Use Cases השלדיים אשר יתכן וחלקם יועברו בשלב הבא לצוות הפיתוח לצורך עיבוי. אורז קבוצות Use Cases לחבילות.
- ארכיטקט תוכנה / מערכת – מסייע בחלוקת המודל לחבילות Use Cases.
- מנהל פרוייקט – מציג שיקולים לחלוקת המודל לחבילות Use Cases השונות.

❖ רשימת תוצרים:

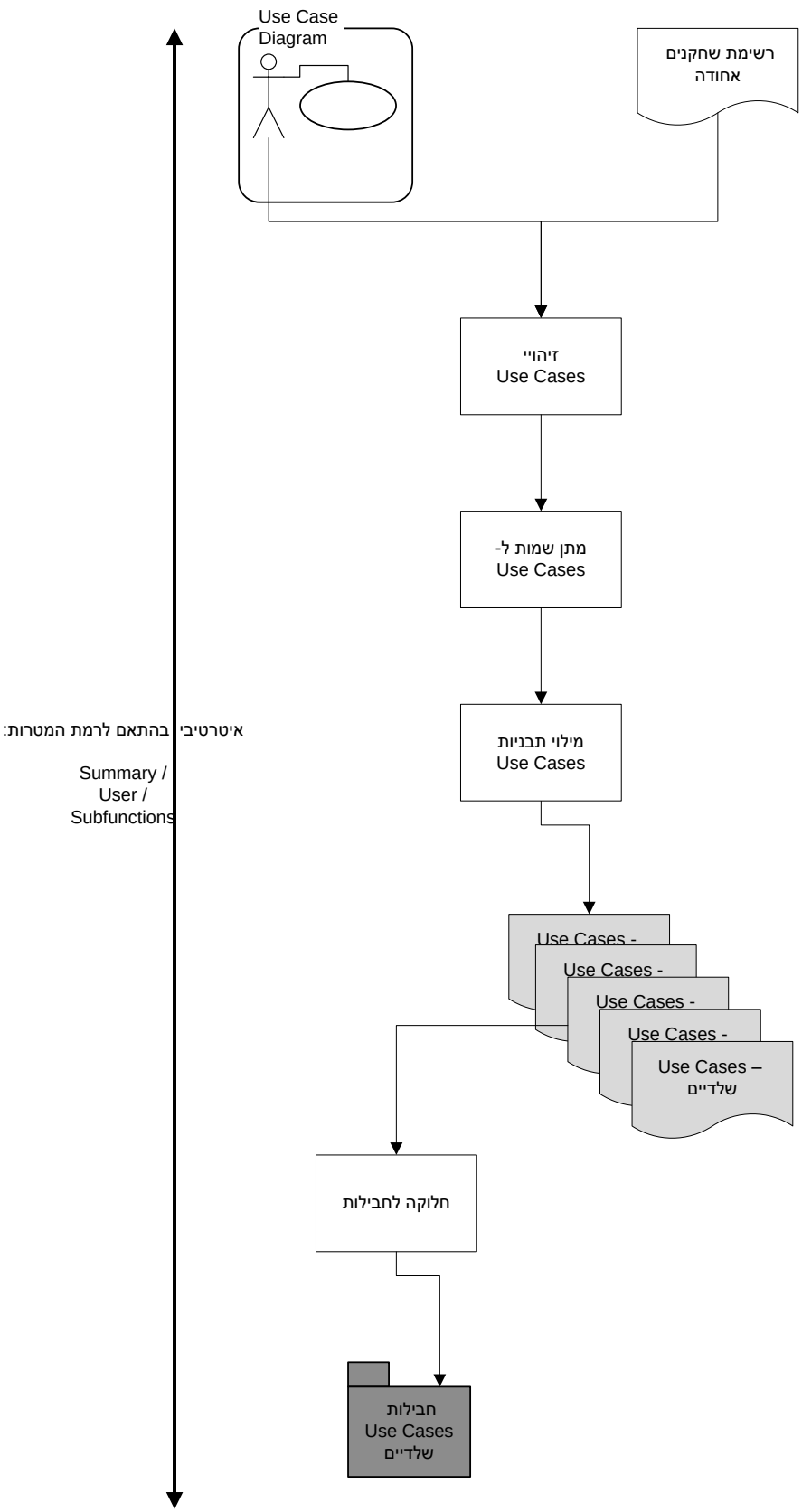
להלן רשימה של התוצרים השונים המופקים בשלב זה:

התוצר	שלם/חלקי	צפי להשלמת התוצר
Use Cases שלדיים	חלקי	שלב הרחבת ה-Use Cases
חבילות Use Cases שלדיים	חלקי	שלב הרחבת ה-Use Cases

טבלה19

08/01/09

להלן תרשים מסכם לשלב III - יצירת Use Cases שלדיים:



6. שלב IV – הרחבת ה- Use Case

<u>רציונל</u>:

מטרות שלב זה הינן להרחיב את ה Use Cases השלדיים אשר נבנו בשלב הקודם. בשלב זה ה"בשר" העיקרי אשר מתווסף לUse Case השלדי הינו הטיפול בתרחישים אלטרנטיביים ובתרחישי כשלון.

ה Use Cases אשר נבחרים לטיפול בשלב זה הינם אותם Use Cases אשר נבחרו בשלב הקודם בהתאם לאיטרציה הנוכחית (Summary-Goals , User-Goals , Subfunctions-Goals).

בתום שלב זה התוצרים כבר יכילו את ההסתעפויות השונות של הUse Cases השונים ולכן נקבל דרישות פונקציונאליות של המערכת ברמת הדיוק הגבוהה ביותר (Cockburn, 2000).

1.6. תוצר נדרש: Use Cases ללא הסתעפויות (תוצר חלקי)

Use Cases ללא הסתעפויות הינם Use Cases מלאים מלבד החלק העוסק בטיפול בהסתעפויות (תרחישים אלטרנטיביים).

1.1.6. <u>המטרה</u>:

הרחבת Use Cases שלדיים לכדי Use Case כמעט מלאים (ללא הסתעפויות).

2.1.6. <u>פעילויות</u>:

❖ בחירת Use Case שלדי והשלמת סעיפים בתבנית:

- נבחר Use Case שלדי מתוך הUse Cases שיצרנו עד כה בהתאם לאיטרציה הנוכחית:
  - איטרציה ראשונה – עיבוי Use Cases שלדיים ברמת Summary-Goals.
  - איטרציה שניה – עיבוי Use Cases שלדיים ברמת User-Goals.
  - איטרציה שלישית – עיבוי Use Cases שלדיים ברמת Subfunctions-Goals.
- נמלא את סעיף בעלי העניין והאינטרסים בתבנית:

יש לציין את בעלי העניין אשר משתתפים בUse Case בו אנו עוסקים. הפעילות כוללת איתור של בעלי העניין הרלוונטיים מתוך רשימת השחקנים האחודה וכן זיהוי האינטרסים שלהם בהקשר של ה Use Case אותו אנו מפרטים. במידת הצורך יש לבצע ראיון חוזר עם בעלי העניין על מנת להבין היטב מהם האינטרסים שלהם אשר רלוונטיים לUse Case בו אנו עוסקים.

> לדוגמא:
>
> אחד מבעלי העניין ב Use Case של משיכת מזומנים מATM הינה חברת האשראי אשר מנפיקה את כרטיסי הכספומט. לחברה אינטרס כי משיכת המזומנים תתבצע באופן תקין וחוקי. ברצונה להגדיל את היקף הנפקות הכרטיסים ובכך להגדיל את הכנסותיה. החברה תרצה להמנע משימוש בלתי חוקי בכרטיסים דבר אשר עשוי לגרום לפגיעה במוניטין ולהוביל לאובדן לקוחות.

- נמלא את סעיפי התנאים: תנאים מוקדמים, תנאים מינימליים, תנאים מובטחים בהצלחה:
  - <u>תנאים מוקדמים</u>: אלו התנאים שמובטח על ידי המערכת כי יתקיימו בטרם הפעלת ה-Use Case. מכיוון שתנאים אלו מובטחים הם לא נבדקים שוב במהלך ריצת ה-Use Case.

לדוגמא:
לקוח ה ATM נכנס למערכת ובוצעה בדיקה כי הוא אכן רשאי להכנס (ללקוח חשבון בנק תקף באחד מסניפי הבנקים המקושרים למערכת).

- <u>תנאים מאוחרים מינימליים</u>: אלו התנאים המועטים אשר מובטחים לבעלי הענין כי יתקיימו גם כאשר המטרה של השחקן הראשי לא מתקיימת. תנאים אלו מתקיימים בכל מקרה בסיום של כל הרצת Use Case גם אם המטרה הושגה וגם אם לאו.

לדוגמא:
מערכת הATM תרשום ביומן האירועים את כל הפעולות אשר בוצעו. גם במקרה של כשלון מובטח לנו כי ישנו קובץ log שניתן לקרוא מתוכו אילו שלבים בוצעו עד לכשלון.

- <u>תנאים מאוחרים בהצלחה</u>: אלו התנאים אשר יתקימו רק במקרה של הרצה מוצלחת של ה-Use Case (בתרחיש ההצלחה הראשי או בתרחיש הצלחה אלטרנטיבי). הרצה מוצלחת היא הרצה שבסיומה המטרה של בעל הענין מושגת ובמקרה זה התנאים המאוחרים בהצלחה מתווספים לתנאים המינימליים אשר הובטחו לנו מראש.

לדוגמא:
מערכת ה ATM תדפיס פתקית עם פרטי הטרנזקציה אשר בוצעה. הפתקית תודפס רק אם הלקוח הצליח לבצע את הפעולה שביקש לבצע (משיכת מזומנים / בירור יתרה וכדי).

לצורך איתור התנאים יש לבחון את הקלטים ל Use Case בו אנו עוסקים ולהבין כיצד ה Use Case מתייחס אליהם. יש להשתמש בקווים מנחים לצורך זיהוי התנאים כפי שמופיע בסעיף "כלים וטכניקות".

- נמלא את סעיף הTrigger. הטריגר מייצג את אותו אירוע אשר גורם להתנעת ה Use Case. במקרים רבים הטריגר הינה פעולה שמבצע השחקן הראשי אך יתכן מקרה כי הטריגר הינו אירוע שמותנה בזמן (לדוגמא: גיבוי אוטומטי יומי). יש לזכור כי יתכן מצב בו יש יותר מטריגר אחד. על מנת להצליח לחשוף את כולם יש לבקש מבעל העניין לתאר את אופן זרימת התהליך. הטריגר בשילוב עם התנאים המקדימים מתאר את מצב המערכת בהתחלת הUse Case. את הטריגר יש לתאר באופן אקטיבי ולא פאסיבי ולנסות להבין מהי רמת האוטומציה הנדרשת בתהליך.

לדוגמא:
עבור ה Use Case של משיכת מזומנים מATM הטריגר: הלקוח מכניס את הכרטיס לATM.

❖ עידון סעיפים קודמים:

בפעילות זו אנו מבצעים סקירה מחודשת של סעיפים שמולאו בUse Case השלדי בשלב הקודם של המודל ומנסים לשפרם במידת הצורך. ההנחה היא שבשלב זה יש לנו הבנה טובה יותר של התמונה ולכן יתכן ונוכל לדייק יותר ו/או לתקן טעויות.

- שם ה Use Case

השם צריך לשקף את מטרתו של השחקן הראשי. לדוגמא: במידה ולאחר חשיבה נוספת וראיונות עם בעלי עניין מתקבלת מסקנה כי השחקן הראשי משתנה, סביר ביותר שיהיה צורך לשנות את שם ה Use Case בהתאם.

- השחקנים המשתתפים

בפעילות זו נבצע סקירה מחודשת של השחקנים המשתתפים ב Use Case. יש לנסות להבין את התמונה הכוללת בתהליך ולדעת "מי עושה מה?" בUse Case הנוכחי. רצוי להעזר בטכניקת "יום בחייו של X" כמפורט בסעיף "כלים וטכניקות".

❖ כתיבת תרחיש ההצלחה הראשי בתצורה מפורטת

בפעילות זו נמיר את תקציר תרחיש ההצלחה שנכתב בשלב גיבוש ה Use Case השלדי לתרחיש הצלחה ראשי מפורט. את תרחיש ההצלחה הראשי נכתוב בתצורת צעדים מפורטת. רצוי ליצור תרשים Use Case אשר נגזר מהתרשים הכללי שיצרנו עבור המערכת בשלב תיחום גבולות המערכת ולפרט אותו בהתאם לפעולות המתרחשות ב Use Case הנוכחי. בתום כתיבת תרחיש ההצלחה הראשי ניתן להסיר את התקציר (Brief Description) מהתבנית של הUse Case.

להלן דוגמא לתרחיש הצלחה ראשי מפורט :

שם ה Use Case : משיכת מזומנים מATM

סקופ : (תכן) ATM

רמה : מטרת משתמש

שחקן ראשי : לקוח ATM

תרחיש הצלחה ראשי:

1) הלקוח מעביר את כרטיס ה ATM בחריץ קורא הכרטיסים.
2) ATM קורא את זיהוי הבנק ואת מספר החשבון.
3) ATM מבקש מהלקוח לבחור שפת ממשק (עברית/אנגלית/ערבית/רוסית).
4) הלקוח בוחר בשפה העברית.
5) ATM מבקש להקיש מספר סודי ולאחריו אישור.
6) הלקוח מקיש את המספר הסודי ולאחריו לוחץ על אישור.
7) ATM מציג ללקוח רשימת פעולות אפשרית לביצוע ע"י הלקוח.
8) הלקוח בוחר בפעולה : "משיכת מזומנים".
9) ATM מבקש מהלקוח להקיש את סכום המשיכה הנדרש במכפלות של 50 ₪ ולבסוף להקיש אישור.
01) הלקוח מקיש את הסכום הנדרש בכפולות של 50 ₪ ולבסוף מקיש על אישור.
11) ATM מעביר למערכת הבנק הראשית דיווח בנוגע לחשבון הלקוח ולסכום שנמשך.
21) מערכת הבנק הראשית מאשרת את המשיכה ומדווחת לATM את היתרה החדשה.
31) ATM מוציא את הסכום הנדרש.
41) ATM שואל את הלקוח האם ברצונו בקבלה.
51) הלקוח משיב בחיוב.
61) ATM מדפיס קבלה עם תיאור סכום המשיכה והיתרה החדשה.
71) ATM מבצע רישום של הטרנזקציה ביומן האירועים.

3.1.6. <u>קלטים</u> :

- ❖ Use Cases שלדיים (תוצר משלב קודם)
- ❖ תרשים Use Case מערכתי (תוצר משלב "תיחום גבולות המערכת")

4.1.6. <u>כלים וטכניקות</u> :

- ❖ שימוש בתבניות Use Cases קיימות.
- ❖ שימוש ברשימת שחקנים אחודה.
- ❖ ראיון עם בעלי עניין :

זהו כלי חשוב מאוד עבור שלב זה. יש להציג בפני בעל העניין שאלות ממוקדות על מנת להצליח לזהות את מרכיבי ה Use Case השונים :

- ? האם תוכל לתאר את אופן זרימת התהליך ?
- ? מה חייב להתבצע בטרם נוכל לבצע זאת ?
- ? מה אתה צריך בטרם תוכל לבצע את הפונקציונליות הזו?
- ? אילו תפקידים מעורבים במשימה זו?
- ? כיצד בעלי התפקידים השונים מתקשרים זה עם זה?
- ? מהי המטרה החשובה ביותר של Use Case זה ?

- ❖ "יום בחייו של השחקן X"

בטכניקה זו אנו מנסים לתאר יום בחייו של כל שחקן. לאחר מכן אנו מתארים את פעילותו ביום האחרון של השבוע , ביום האחרון של החודש , ביום האחרון של הרבעון וכך הלאה. בעזרת טכניקה זו אנו מצליחים לחשוף קשרים ויחסי גומלין אשר יתכן ולא הצלחנו לראות קודם לכן.

5.1.6. <u>איכות</u>:

- ❖ קווים מנחים עבור תנאים מוקדמים ומאוחרים (Leffingwell and ,2005 Widrig)
  - המצבים המתוארים ע״י התנאים צריכים להיות ניתנים להבחנה (observable) ע״י המשתמש. לדוגמא: ״המשתמש ביצע כניסה למערכת״.
  - תנאי מקדים הינו אילוץ עבור אפשרות התחלת ה Use Case. תנאי מקדים איננו אירוע אשר מתחיל את ה Use Case.
  - תנאים מקדימים ומאוחרים מוחלים על כל תרחישי ה Use Case אלא אם צויינו עבור תרחיש מסויים.
  - תנאי מאוחר (מינימלי) עבור Use Case צריך להתקיים בכל מקרה עבור כל התרחישים האלטרנטיביים ולא רק בעבור התרחיש הראשי.
  - תנאי מאוחר (בהצלחה) יכול לשמש ככלי לתיאור ה Use Case , לאחר כתיבתו נתאר את הצעדים ההכרחיים להשגתו.

08/01/09

❖ לצורך אימות של תבנית הUse Case נשתמש בטבלת Pass/Fail. במידה ומתגלות בעיות , יש לבצע תיקון ולהעריך את השפעתו על Use Cases אחרים.

להלן טבלת Pass/Fail עבור Use Case בודד (Cockburn, 2000):

השדה בתבנית ה Use Case	שאלות הבקרה:
שם ה Use Case	בוצע בשלב קודם – "גיבוש Use Case שלדי"
סקופ	בוצע בשלב קודם – "גיבוש Use Case שלדי"
רמה	בוצע בשלב קודם – "גיבוש Use Case שלדי"
שחקן ראשי	בוצע בשלב קודם – "גיבוש Use Case שלדי"
תנאים מקדימים	1. האם הם הכרחיים והאם המערכת יכולה להבטיח קיומם? 2. האם באמת ה Use Case לא בודק את התקיימותם?
בעלי עניין	3. האם ישנה התייחסות לסעיף זה?
תנאים מאוחרים	4. במידה והם קיימים האם האינטרסים של כל בעליי העניין נשמרים?
תרחיש הצלחה ראשי	5. האם התרחיש רץ החל מהטריגר ועד להגעה אל תנאי סיום מוצלח? 6. האם רצף הפעולות נכון? 7. האם התרחיש מכיל בין 3 ל 9 צעדים?
לכל צעד בתרחיש	8. האם הוא מבוטא כמטרה אשר הושגה בהצלחה? 9. האם התהליך מתקדם לאחר הצלחה של הצעד? 01. האם זה ברור איזה שחקן מבצע את המטרה? ("מי מחזיק במושכות?") 11. האם כוונת השחקן ברורה? 21. האם רמת המטרה של הצעד נמוכה במעט מרמת המטרה הכוללת של ה Use Case ? 31. האם אתה בטוח כי הצעד לא מתאר את ממשק המשתמש של המערכת? 41. האם זה ברור איזו אינפורמציה מועברת?
הרחבות והסתעפויות	יבוצע בסעיף האיכות של התוצר הבא
כללי	יבוצע בסעיף האיכות של התוצר הבא

טבלה20

6.1.6. <u>סיכונים ושגיאות נפוצות</u> :

קטגוריה	שגיאה נפוצה
תנאים מוקדמים	ביצוע בדיקות מיותרות במהלך ה Use Case לתנאים אשר מופיעים כתנאים מוקדמים.
תנאים מאוחרים	פירוט מיותר של פעולות במהלך ה Use Case לתנאים אשר מובטחים בכל מקרה בסיום ההרצה.
טריגר	תיאור אירוע שאיננו גורם להתנעת ה Use Case הנוכחי.
תרחיש הצלחה ראשי	תיאור פעולות אשר אינן חלק מתרחיש ההצלחה הראשי אלא שייכות להסתעפויות שונות.
צעדי התרחיש הראשי	תיאור של צעד בתרחיש אשר איננו גורם להתקדמות בתהליך ואיננו מתאר השגת מטרה כלשהי בצורה מוצלחת.

טבלה21

2.6. תוצר נדרש : Use Cases מלאים (תוצר מלא כולל הסתעפויות)

1.2.6. <u>המטרה</u> :

השלמת ה Use Cases לכדי Use Case מלאים (כולל הסתעפויות).

2.2.6. <u>פעילויות</u> :

❖ הפקת רשימת תרחישי הסתעפות [13] :

פעילות זו כרוכה בביצוע של סיעור מוחות לצורך יצירת רשימה של הרחבות. הרחבות אלו הסתעפויות מתוך תרחיש ההצלחה הראשי אשר יתארו כשלונות או תרחישי הצלחה אלטרנטיביים. בכל נקודה שבה ההתנהגות עשויה להתפצל בעקבות תנאי מסויים יש לרשום את התנאי ואת הצעדים לטיפול בו. הרחבות רבות מתמזגות בהמשך חזרה עם תרחיש ההצלחה הראשי. יש להעזר בקווים מנחים לצורך ביצוע הפעילות.

להלן דוגמא לרשימת תרחישי הסתעפות עבור Use Case של משיכת מזומנים :

1. קורא כרטיסים לא תקין או כרטיס שחוק.
2. כרטיס השייך לבנק שאיננו משוייך לרשת הבנקים הנתמכים.
3. הוכנס מספר סודי שגוי.
4. הלקוח לא הקיש את המספר הסודי בזמן.
5. ATM איננו פעיל.
6. רשת המחשבים אליה ה ATM מתחבר למטה.
7. אין מספיק כסף בחשבון הלקוח.
8. הלקוח לא הזין את הסכום המבוקש בזמן.
9. הלקוח הזין סכום מבוקש שאיננו בכפולות של 50 ₪ .
01. הסכום המבוקש גדול מהסכום המירבי האפשרי.
11. מחשב הבנק או הרשת נופלים תוך כדי ביצוע הטרנזקציה.
21. אין מספיק כסף במכונת ה ATM.
31. שטרות הכסף נתקעים תוך כדי הוצאתם.
41. נייר המדפסת נגמר או נתקע.
51. הלקוח איננו לוקח את השטרות שהוצאו עבורו.

[13] תרחישים אלטרנטיביים ותרחישי כשלון

❖ פירוט תרחישי ההסתעפות :

בפעילות זו מתבצע מעבר על רשימת ההרחבות ומתבצעת כתיבה מפורטת של כל הרחבה בנפרד. תרחישי ההסתעפות נכתבים בצורה מקוננת בהתאם לסוג התבנית אשר נבחרה. להלן דוגמא להסתעפות (מספר 3 ברשימת הקודמת, התפצלות מצעד מס' 6 ב Use Case של משיכת מזומנים) :

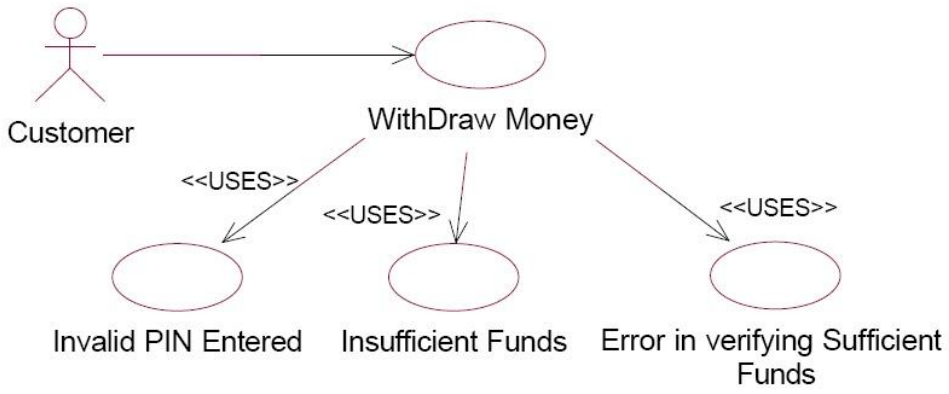
1.6 הלקוח מקיש מספר סודי שגוי.
1.1.6 הATM מודיע ללקוח כי המספר שהוכנס שגוי.
2.1.6 הATM פולט החוצה את הכרטיס.
3.1.6 הATM מעביר דיווח למחשב הבנק אשר רושם את פרטי הפעולה שנכשלה.

❖ יצירת תרשימים :

פעילות זו הינה פעילות רשות. התרשימים מיועדים בעיקר עבור מצגות והצגת Use Cases עבור דרג ניהולי. את התרשימים רצוי ליצור עבור Use Cases עיקריים ובד"כ עבור Use Cases ברמת Summary או User. לעיתים ניצור תרשים על מנת להדגיש חלק מסוים ב Use Case. התרשימים יצורפו לתבנית ה Use Case כחלק אינטגרלי של ה Use Case. תרשימי ה UML המקובלים עבור Use Case Model הינם :

- תרשים Use Case (כמפורט בנספח – תרשימי Use Cases) :

לדוגמא : דיאגרמת Use Case עבור משיכת מזומנים <u>בדגש על הסתעפויות</u> :



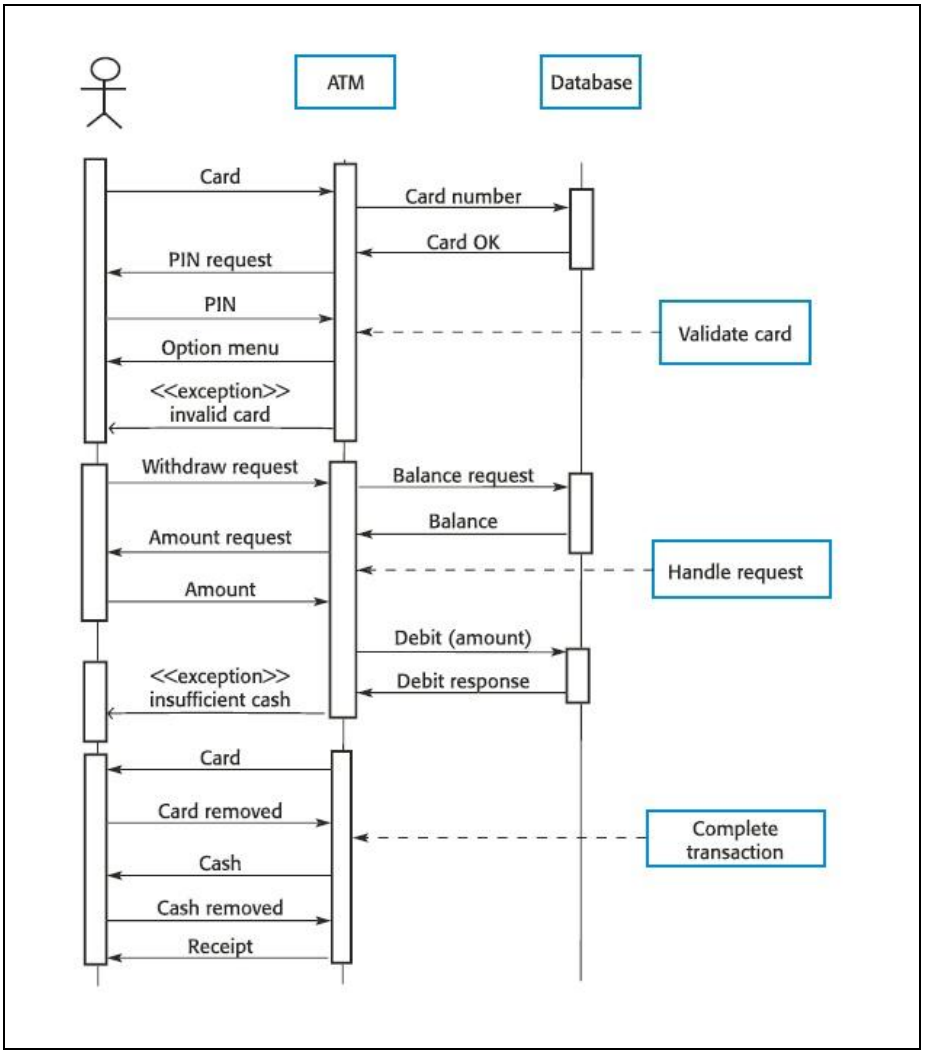
- <u>דיאגרמת Sequence</u>

דיאגרמת Sequence הינה דיאגרמה טובה לתיאור סדר ורצף הפעולות של תרחישי ה Use Case. הדיאגרמה הינה דיאגרמה תקנית של UML. דיאגרמת Sequence בסיסית לתיאור Use Case תכיל :

- אובייקטים המופיעים כמלבנים ובתוכם שם האובייקט (מייצגים את השחקנים השונים והמערכת המנותחת).
- קווים אנכיים מקבילים (מייצגים את קו-החיים של האובייקטים השונים).
- Linkים - חצים אופקיים וכיתוב מעליהם (מייצגים העברת הודעות בין האובייקטים השונים).

לצורך יצירת התרשים יש להעזר בקווים מנחים ליצירת תרשים Sequence כמפורט בהמשך.

להלן דוגמא לדיאגרמת Sequence המתארת Use Case של משיכת מזומנים מ ATM :



3.2.6. <u>קלטים</u>:

- ❖ Use Cases ללא הסתעפויות (תוצר קודם)
- ❖ תרשים Use Case מערכתי (תוצר משלב "תיחום גבולות המערכת")

4.2.6. <u>כלים וטכניקות</u>:

- ❖ סיעור מוחות – לצורך יצירת רשימת הסתעפויות לכל Use Case.
- ❖ שימוש בדיאגרמות Use Case – בעיקר לצורך הדגשת ההסתעפויות השונות.
- ❖ שימוש בקווים מנחים עבור יצירת תרשימי Sequence המתארים Use Cases (Rosenberg and Scott ,2001):
  - יש להבין תחילה מהי מטרת התרשים על מנת להפיק את המירב ממנו.
  - יש ליצור תרשים אחד גם עבור תרחיש ההצלחה הראשי וגם עבור ההסתעפויות של אותו ה Use Case.
  - התרשים צריך לשקף את התנהגות המערכת בUse Case המתואר וכיצד מטרת השחקן מושגת.
  - ההודעות המועברות בין האובייקטים צריכות להגזר מתוך צעדי התרחשי של ה Use Case.
  - התמקד באינטרקציות העיקריות והקריטיות.
  - השתמש בכלי ליצירת תרשימים התומך ב UML תקני.
  - השתמש בתרשים בהמשך לצורך יצירת מחלקות והפעולות שלהן.
  - במידה וקיים מודל סטטי – וודא כי התרשים שהינו חלק מהמודל הדינמי עקבי ולא סותר את המודל הסטטי.
- ❖ שימוש בקווים מנחים עבור הסתעפויות (Cockburn, 2000):
  - להלן נקודות מפתח לזיהוי הסתעפויות:
    - מסלול הצלחה אלטרנטיבי (לקוח בוחר במשיכת מזומנים מהירה ללא הדפסת קבלה)
    - השחקן הראשי מתנהג בצורה שגויה (הלקוח מזין קוד סודי שגוי)
    - השחקן הראשי לא מבצע אף פעולה בזמן שמצופה ממנו (חלף הזמן המקסימלי והלקוח לא הזין קוד סודי)
    - כל צעד ובו פסקה כגון "המערכת מבצעת וידוא ש..." יכיל הסתעפות ובה מצב בו הוידוא נכשל (ללקוח אין מספיק כסף בחשבון)
    - חוסר תגובתיות של שחקן משני (רשת המחשוב נפלה)
    - תקלה פנימית במערכת שלנו שיש לטפל בה כחלק מההתנהלות השוטפת (שטרות נתקעו בדרך החוצה)

- תקלה פנימית במערכת שלנו המשליכה על שחקנים נוספים (טרנזקציה נכשלה תוך כדי ביצועה)
- תקלת ביצועים קריטית שיש לזהות ולטפל בה (אישור הכרטיס נארך זמן הגדול מ 10 שניות).

- יש לרשום את ההסתעפות מנקודת מבט המאפשרת גילוי על ידי המערכת. לדוגמא: אין לרשום "הלקוח שכח את הקוד הסודי" אלא "הלקוח לא הזין את הקוד הסודי בזמן" או "הלקוח הזין קוד סודי שגוי".
- בסיעור המוחות יש ליצור רשימה עם כל ההסתעפויות האפשריות גם אם ישנו חשד לחפיפה. את הרשימה יש לצמצם בהמשך ע"י מיזוג הסתעפויות ו/או מחיקת הסתעפויות.
- הסתעפויות אשר המערכת לא צריכה לטפל בהם יש למחוק מהרשימה. (לדוגמא: הלקוח שכח להביא עמו את כרטיס ה ATM). הסתעפות שתשאר ברשימה תענה על 2 דרישות:
  - המערכת חייבת לזהות את התנאי להסתעפות
  - המערכת חייבת לטפל בתנאי להסתעפות

  במידה וישנה הסתעפות שלא ניתנת לזיהוי אך ניתן להמירה – נבצע זאת ("הלקוח שכח את הקוד הסודי" תומר ל"הלקוח לא הזין את הקוד הסודי בזמן").
- יש למזג הסתעפויות שקולות. המיזוג יתבצע בהתאם לנקודת מבט המערכת ואופן טיפולה במקרים השונים. אם הטיפול הוא זהה יש למזג. לדוגמא: "קורא הכרטיסים לא תקין" , "כרטיס הלקוח שחוק" , "הכרטיס שהוכנס איננו כרטיס ATM" כולם יטופלו באופן זהה ע"י המערכת.
- הסתעפויות יש לרשום בהתאם לרמת ה Use Case המטופל. יתכן ו Use Case ברמה גבוהה מכיל קריאה ל Use Case ברמה נמוכה יותר כאחד מצעדיו. ברמה הגבוהה אנו נרשום הסתעפות עבור כשלון ה Use Case ברמה הנמוכה ללא פירוט של כל אפשרויות הכשלון כפי שיפורטו כהסתעפויות ב Use Case ברמה הנמוכה. לדוגמא: Use Case ברמת משתמש של "מילוי כסף במכונת ATM" קורא באחד מצעדיו ל Use Case ברמת Subfunction של "איפוס מונים ב ATM" המכיל תרחישי הסתעפות רבים המציינים כשלונות וטיפול שונה בכ"א מהם. עבור ה Use Case ברמת המשתמש נציין רק הסתעפות שמציינת כי איפוס המונים ב ATM נכשל ולא נציין את כל הסיבות האפשריות.

5.2.6. <u>איכות</u> :

❖ אימות דיאגרמות Use Case :

- יש לוודא כי תרשים ה UML הינו תקני ובהתאם לסינטקס של UML.
- יש לוודא כי בכל תרשים יש לפחות שחקן אחד המקושר לבועת Use Case אחת לפחות.
- יש לוודא כי השחקן אכן חיצוני למערכת ואיננה חלק ממנה (כגון בסיס-נתונים פנימי).
- אלמנטים הקרובים זה לזה מבחינה סמנטית יש לצייר קרובים זה לזה פיזית.
- יש לצמצם למינימום את מספר הקווים אשר חותכים זה את זה.
- יש לתת שם לדיאגרמה בהתאם לשם ה Use Case .
- יש לבדוק האם יש צורך להוסיף בדיאגרמה שימוש ב Use Case אחר (קשר include).
- יש לבדוק האם ה Use Cases המתואר בדיאגרמה הוא למעשה הרחבה של Use Case אחר (במקרה זה , יש להוסיף קשר extends).

❖ אימות תרשים Sequence המתאר Use Case :

- כל האובייקטים בתרשים יופיעו כשחקנים (או כמערכת הממודלת) מתוך תבנית ה Use Case ותרשים ה Use Case במידה וקיים (עקיבות).
- הודעות בין אובייקטים שונים יועברו אך ורק כחלק מ Link.
- הודעה תהיה חלק משם פעולה (כמתואר בצעד של תבנית ה Use Case).
- במידה ומצוין ערך חוזר בדיאגרמה יש לוודא כי נעשה בו שימוש.

❖ אימות הסתעפויות – להלן המשך טבלת Pass/Fail אשר הופיעה בתוצר קודם :

הרחבות והסתעפויות	51. האם המערכת מסוגלת לזהות הסתעפות זו? 61. האם המערכת חייבת לטפל בהסתעפות זו?
כללי	71. יש להציג את הUse Case לשחקנים ולשאול "האם זה מה שרציתי?" 81. יש להציג את ה Use Case למפתחים ולשאול "האם תוכל לממש את זה?"

טבלה22

6.2.6. <u>סיכונים ושגיאות נפוצות</u> :

השגיאה	**מה עשוי להגרם / הנחיות**
תרשים Use Cases מורכב ממספר רב של אלמנטים , קווים חותכים זה את זה.	התרשים איננו ברור. אין לנסות להכניס כל פרט ופרט המתואר בUse Case לתוך התרשים. התרשים צריך לתאר את עיקרי הפעילויות ולהדגיש פרטים מסוימים.
רישום של הסתעפות אשר המערכת אינה צריכה לטפל בה .	הUse Case מתנפח ללא צורך והופך לפחות קריא וברור.
ה Use Case מכיל 2 או יותר הסתעפויות אשר שקולות זו לזו.	הUse Case מתנפח ללא צורך והופך לפחות קריא וברור. יש למזג הסתעפויות שקולות בהתאם לאופן הטיפול במקרים השונים ע״י המערכת.
ה Use Case מכיל הסתעפויות ברמה הנמוכה מרמת הUse Case .	נוצר חוסר עקביות במודל. פירוט ההסתעפויות יבוצע בהתאם לרמת ה Use Case.
ישנן הסתעפויות אשר מופיעות כחלק מתרחיש ההצלחה הראשי.	תרחיש ההצלחה הראשי הו]ך לבלתי ברור. יש לטפל בהסתעפויות בנפרד וכלל לא להזכירן בתרחיש ההצלחה הראשי.

טבלה23

3.6. תוצר נדרש: חבילות Use Cases (תוצר מלא)

חבילות Use Cases מלאות הינן חבילות המכילות אוסף של Use Cases שלמים בעלי מכנה משותף.

1.3.6. המטרה:

בשלב זה יש בידנו Use Cases מלאים בהתאם לרמה של האיטרציה המטופלת. המטרה הינה לבצע אירגון טוב יותר של ה Use Cases שנכתבו עד כה ולמנוע "התפוצצות מצבים" (Cockburn, 1997). מניעה של "התפוצצות מצבים" מתבצעת בשתי רמות – ברמה הראשונה ישנה כתיבת תרחישי ההסתעפויות כחלק מ Use Case בודד. ברמה השנייה מבוצע ארגון של מספר Use Cases (איחוד, איגוד לחבילות). ארגון המודל מאפשר ניהול ושליטה טובה יותר על ה Use Cases בחתכים שונים. לדוגמא: חלוקה לחבילות מאפשרת לחלקם לצוותי פיתוח שונים ואף להציג את המודל ללקוח במבט על בצורה ברורה. בשלב זה אנו מבצעים "נקיון" ואף מטפלים במקרים מיוחדים.

2.3.6. פעילויות:

- שבירת Use Case מפורט למספר Use Cases:

בפעילות זו אנו מאתרים Use Cases "כלליים מדיי" ומפרקים אותם למספר Use Cases. פעילות זו יש לבצע לאחר שיש בידינו קבוצת Use Cases מורחבים. פעילות זו מעלה שאלות כגון: מהו "הגודל" המומלץ עבור Use Case? , מהי הנקודה בה מומלץ לבצע את השבירה של ה Use Case? וכו'. פעילות זו יש לבצע ביתר זהירות מכיוון שבאופן בסיסי פירוק אלמנטים שוב ושוב עד להגעה לאלמנט אטומי זו פעילות המתאימה יותר לתפיסה של חשיבה מבנית ופונקציונלית. לצורך ביצוע פירוק נכון יש להשתמש בטכניקה של "Granularity" כמתואר להלן:

- Use Case Granularity –

זהו הScope היחסי של Use Case מסויים בהשואה לScope של האפליקציה כולה. Granularity זהו מדד יחסי ולא קימות מטריקות מדוייקות לקבוע את ה Granularity של כל פרוייקט. הרעיון הוא להגיע לUse Cases אשר פחות או יותר "שווים בגודלם" זה לזה. פרמטרים מקובלים לצורך השוואה -

08/01/09

- זמן איטרציות – האם ה Use Case ניתן למימוש בסדר גודל של זמן האיטרציה? (נניח : 3 שבועות) במידה וההערכת זמנים עבורו גדולה בהרבה מזמן האיטרציה אז ה Use Case נמצא כמועמד לפירוק. במידה והערכת זמנים עבורו קטנה בהרבה מזמן האיטרציה אז ה Use Case נמצא כמועמד לאיחוד עם Use Case אחר.
- האם אנליסט יחיד מסוגל לעבוד על ה Use Case ? במידה והתשובה היא לא אז הUse Case הוא מועמד לפירוק.

❖ איחוד מספר Use Cases לכדי Use Case בודד :

בפעילות זו אנו מאתרים מספר Use Cases "קטנים" אשר חיבורם יהווה Use Case יחיד בעל Scope מתאים. אנו מחפשים למעשה Use Cases שבאופן טבעי היו אמורים להכתב כUse Case יחיד. פעילות זו אינה מחייבת מציאת Use Cases דומים ואיחודם. לצורך כך יש להעזר בטכניקת Use Case Granularity.

❖ <u>טיפול ב Use Cases מסוג CRUD</u> :

באותו האופן שבוצע בשלב קודם "גיבוש Use Case שלדי" .

❖ <u>אריזת Use Cases לחבילות</u> :

באותו האופן שבוצע בשלב קודם "גיבוש Use Case שלדי".

3.3.6. <u>קלטים</u> :

❖ Use Cases מלאים (כולל הסתעפויות).

4.3.6. <u>כלים וטכניקות</u> :

❖ שימוש בטכניקת Granularity לצורך פירוק Use Cases.

5.3.6. <u>איכות</u>:

❖ שימוש בשאלות מנחות (התשובה צריכה להיות <u>כן</u> עבור כל השאלות):

? האם כל ה Use Cases מאורגנים בחבילות? האם ישנו תיעוד המציין את המפתח על פיו בוצעה החלוקה לחבילות?

? האם Use Cases "קטנים" אוחדו לUse Case יחיד ?

? האם קיים תרשים חבילות?

? האם הייתה התייחסות ל Use Cases מסוג CRUD?

? האם ישנה לכידות בין ה Use Cases השונים באותה החבילה ?

? האם אין תלות מעגלית בין החבילות ?

? האם שם החבילה משקף את המשותף לUse Cases באותה החבילה ?

6.3.6. <u>סיכונים ושגיאות נפוצות</u> (Kulak and Guiney ,2000):

השגיאה	**מה עשוי להגרם / הנחיות**
אריזת Use Cases לחבילות בצורה לא טובה.	איגוד לחבילות חייב להתבצע ע"פ קריטריון משותף וכזה שבעלי העניין ימצאו בו הגיון. האיגוד יבוצע באמצעות דיאגרמת חבילות של UML ע"מ לאפשר הצגה ברורה.
שימוש בקריטריון Granularity שונה בעבור איחוד ופירוק Use Cases.	יתקבל מודל המכיל חבילות לא עקביות. יש לבצע איחוד ופירוק של Use Cases ע"פ אותו המפתח (שימוש בטכניקת Granularity).

טבלה24

08/01/09

סיכום:

שלב זה כקודמו מתבצע בהתאם לאיטרציה בה אנו נמצאים (Summary- Use Cases Subfunctions-Goals,User-Goals,Goals). בשלב זה אנו משלימים את ה Use Cases השלדיים אשר נוצרו בשלב הקודם. אנו מבצעים סקירה מחודשת של עיקרי הנושאים שנקבעו עד כה (שם ה Use Case , השחקנים המשתתפים), מפרטים את תרחיש ההצלחה הראשי שתואר בקצרה ומשלימים את יתר הסעיפים (תנאים , הרחבות וכו'). התהליך כולל יצירת שלב ביניים ובו מתקבל מודל Use Cases ללא טיפול בהסתעפויות. באופן זה ניתן לקבל תמונה ברורה בטרם אנו צוללים ומעמיקים בכל Use Case. כמו כן, התהליך מאפשר רמה נוספת של איטרטיביות מעבר לחלוקה הראשונית של איטרציות ע"פ רמת המטרות של הUse Cases. בהמשך שלב זה מתבצע טיפול בהסתעפויות לצורך השלמת ה Use Cases ולבסוף אנו מבצעים ארגון מבני של ה-Use Cases ומחלקים לחבילות.

❖ בעלי תפקידים:

- מנתח דרישות – משלים את ה- Use Cases השלדיים לכדי Use Cases מלאים, מראיין את בעלי העניין, מבצע איחוד ופירוק של Use Cases ע"פ הנדרש. מבצע חלוקת המודל לחבילות Use Cases. מבצע אימות של Use Cases אשר כתב.
- ארכיטקט תוכנה / מערכת – מסייע בחלוקת המודל לחבילות Use Cases.
- מנהל פרוייקט – מציג שיקולים לחלוקת המודל לחבילות Use Cases השונות.

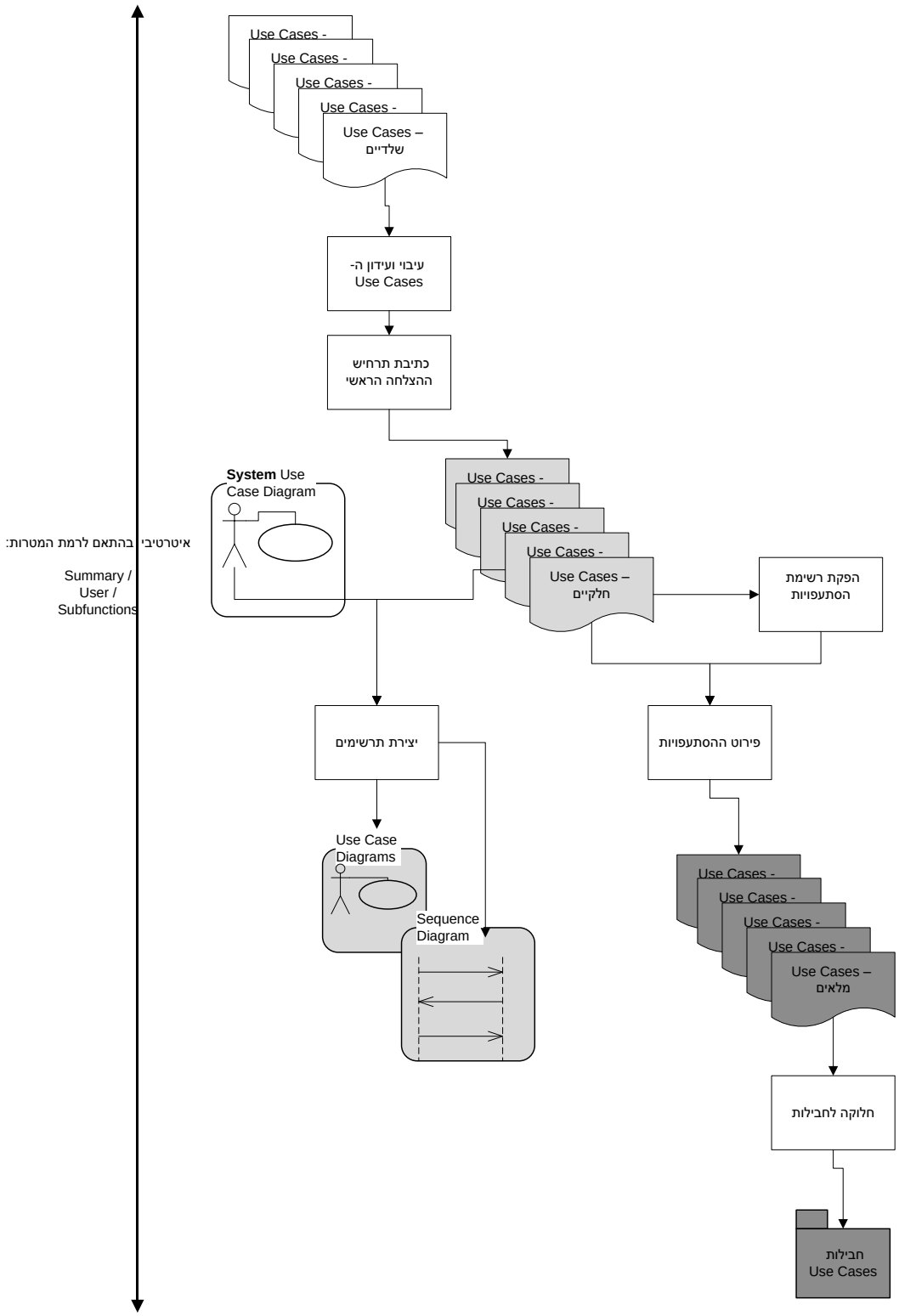
❖ רשימת תוצרים:

להלן רשימה של התוצרים השונים המופקים בשלב זה:

התוצר	שלם/חלקי	צפי להשלמת התוצר
Use Cases ללא הסתעפויות	חלקי	בשלב הנוכחי
Use Cases מלאים (כולל הסתעפויות)	שלם	
חבילות Use Cases	שלם	

טבלה25

להלן תרשים מסכם לשלב IV - הרחבת ה Use Case :



7. שלב V – הבטחת איכות המודל

<u>רציונל</u>:

בשלב זה אנו רוצים להבטיח כי מודל ה Use Cases אשר בנינו עומד בסטנדרטים גבוהים של איכות. בשלב זה יש לבצע תיקוף של מודל ה Use Cases. השלב מכיל פעילויות שונות של הבטחת איכות אשר חלקן מבוצעות בתום בניית המודל ואת חלקן יש לבצע במהלך בניית המודל לאורך כל שלבי המתודולוגיה. כאשר יש בידנו מודל איכותי ומתוקף נרצה לשלבו בתהליך הפיתוח הכללי ולכן שלב זה מכיל נדבך נוסף - שילוב המודל בארכיטקטורה הכללית של המערכת.

הבטחת איכות המודל נותנת לנו מדדים להעריך גם את איכות הדרישות המאופיינות של המערכת. ככל שתוצרי המודל יהיו איכותיים יותר כך המערכת הסופית שתתקבל תהיה איכותית יותר.

כפי שצוין בפרק המבוא המתודולוגיה המוצגת בעבודה זו איננה מיוצגת בייצוג פורמאלי ועל כן איננה כוללת מדדי הבטחת איכות כמותיים אלא <u>מדדים איכותיים בלבד</u>.

על אף הנאמר לעיל , המתודולוגיה מכילה פעילויות הבטחת איכות רבות אשר ביצוע דקדקני שלהן יבטיח במידה רבה את איכות התוצרים הנדרשים.

1.7. תוצר נדרש: מודל Use Case מתוקף

מודל Use Case מתוקף הינו מודל שעבר וריפיקציה בהתאם למבדקי איכות שנקבעו מראש.

1.1.7. <u>המטרה</u>:

תיקוף (Validation) של מודל ה Use Case .

2.1.7. <u>פעילויות</u>:

- תיקוף בהתאם לטבלת בדיקות (Anda and Sjoberg, 2002) – פעילות זו ניתנת לביצוע בתום כל איטרציה בשלב בו ישנם Use Cases מלאים.

קטגוריה	בדיקות
שחקנים	האם ישנן יישויות חיצוניות נוספות איתן המערכת מתקשרת אך לא יוצגו בעזרת אף שחקן במודל?
	האם המודל מכיל שחקנים מיותרים אשר אינם מספקים או מקבלים מידע מהמערכת?
	האם כל השחקנים תוארו באופן ברור והאם במבט לאחור המטרות שתוארו עבורם אכן משקפות את המטרות שלהם?
	האם ברור אילו שחקנים משתתפים בכל Use Case? האם זה ניתן להבנה באופן קל מתוך תבנית ודיאגרמת ה Use Case? האם השחקנים מחוברים לUse Case המתאים?
Use Cases	האם ישנה פונקציונליות של המערכת שאיננה מכוסה ע"י אחד מה Use Cases? האם ישנן מטרות נוספות לחלק מהשחקנים אשר לא תוארו באף Use Case ?
	האם ישנם Use Cases מיותרים אשר: נמצאים מחוץ לגבולות המערכת? לא מספקים מטרה של שחקן? מהווים כפילות של Use Case אחר במערכת?
	האם כל ה Use Cases מספקים את המטרה אשר מופיעה בשם ה Use Case?
	האם ישנה עקביות בין תיאור השחקן ומטרותיו ברשימת שחקן-מטרה לבין תיאורו כפי שמופיע ב Use Cases בהם הוא משתתף?
	האם זה מספיק ברור מתוך ה Use Case כיצד מושגת המטרה?
קשרים בין Use Cases שונים	האם ישנה התאמה בין דיאגרמת הUse Case לבין תיאורה המילולי (תבנית)? האם הקשרים המופיעים בדיאגרמה באים לידי ביטוי בתבנית ?
	האם בוצע שימוש נכון בקשר מסוג Include או שמא הקשר איננו מתאים לתיאור היחס בין ה Use Cases ?

האם התנהגות ה Use Case מתנגשת עם התנהגות של Use Case אחר?	
האם Use Cases הקשורים זה לזה נמצאים באותה רמת פירוט ?	

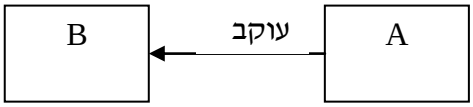
טבלה26

התיקוף נחתם פורמלית לאחר ביצוע סקר תיקוף בהובלת מנתח הדרישות ובהשתתפות ארכיטקט התוכנה , מנהל הפרויקט וצוותי הפיתוח.

- יצירת מטריצת עקיבות :

עקיבות (Traceability) של דרישות זו טכניקה מוכחת אשר מגדילה את איכות ואמינות המערכת. עקיבות שזורה לאורך התהליך כולו ועוברת דרך זיהוי הדרישות , ה Use Cases ולבסוף דרך היישום והבדיקות. עקיבות זו טכניקת מפתח בניהול ה Use Cases. מודל ה Use Cases צריך להיבנות בצורה כזו שיהיה גמיש לשינויים אשר יתרחשו. עקיבות מסייעת לנתח את השפעת השינויים על המודל ועל יתר חלקי הפרוייקט.

קשר עקיבות הוא סוג של קשר תלות בין אלמנטים :



איור16

לקשר התלות ישנו כיוון. אלמנט A (נניח Use Case) עשוי להשפיע על אלמנט B (נניח Test Case) אבל בכיוון ההפוך התלות איננה הכרחית.

על מנת לנהל את ה Use Cases והשפעת השינויים בהם נחזיק מספר מטריצות של עקיבות. מומלץ לנהל את ה Use Cases בכלי לניהול דרישות (כגון Rational Requisite Pro) אשר מאפשר ייצור אוטומטי של מטריצות עקיבות.

להלן דוגמא למטריצת עקיבות :

	UC1	UC2	UC3	UC4	UC5
TC1		↑		↑	↑
TC2			↥		
TC3					
TC4	↑				
TC5		↑	↥		

איור – 17 מטריצת עקיבות בין Test Cases לבין Use Cases

באיור הנ״ל ניתן לראות דוגמא למטריצת עקיבות המשקפת את התלות בין מספר Use Cases לבין מספר Test Cases המיוצגים בעזרת זיהוי ייחודי. Test Cases נועדו לצורך בדיקת המערכת בהתאם לדרישות. במידה והדרישות מיוצגות בעזרת מודל Use Case כמתואר במתודולוגיה הנוכחית אז כדאי לחולל Test Cases מתאימים באמצעות טכניקה פשוטה יחסית (Gutierrez, J. , Escalona, M. and Torres, L.,2006). האלמנטים מנוהלים בעזרת כלי לניהול דרישות כגון Rational Requisite Pro. חץ (↑) מסמן כי ישנו קשר תלות של עקיבות בין אלמנט מסויים (למשל TC4) לבין אלמנט אחר (למשל UC1). חץ עם קו תחתי (↑) מסמן ״חשד״ - אלמנט מסויים השתנה ויש לבדוק האם נדרש עדכון של האלמנט התלוי בו.

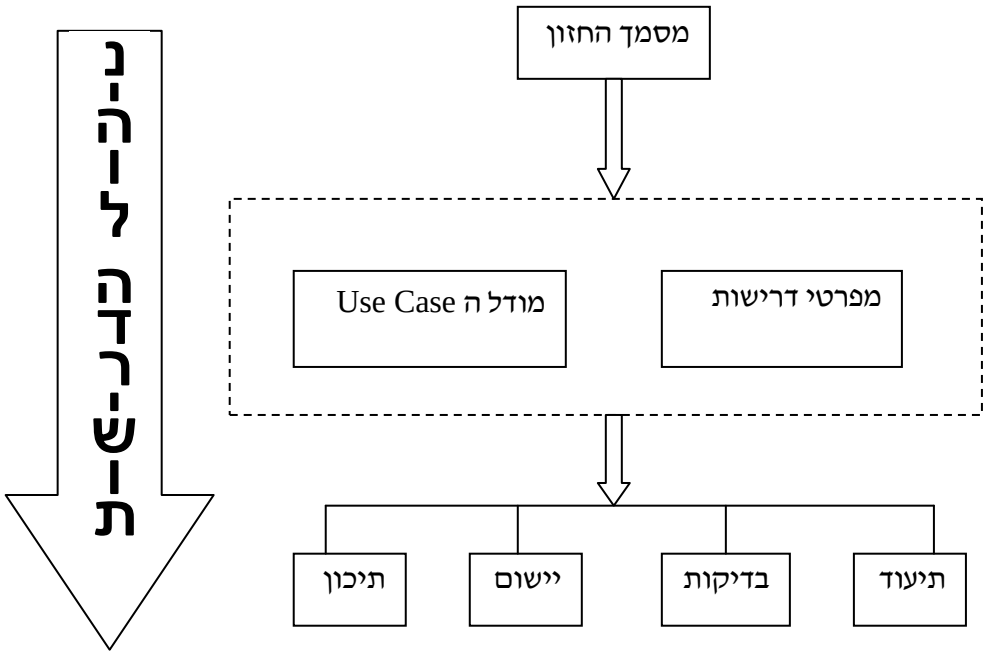
❖ ניהול שינויים במודל ה Use Case :

בפעילות זו ננהל את השינויים באלמנטים מתוך המודל או אלמנטים המשפיעים על חלקים במודל. פעילות זו תתבצע באמצעות תהליך מובנה (Leffingwell and Widrig ,2005) :

- וידוא כי השינוי בלתי נמנע ותכנון עבורו.
  בתת-שלב זה יש לוודא כי השינוי הכרחי ולא ניתן לוותר עליו. שינויים לגיטימיים הם כאלו שהמקור שלהם הוא אחד מבעלי העניין. אין לפחד מהכנסת השינויים אך יש צורך להבין את המשמעות הכרוכה בהם ולבצע זאת באופן מסודר עם ״עיניים פקוחות״ לאורך כל הדרך.
- מתן קו-בסיס (Baseline) לדרישות במודל.
  בכל איטרציה יש לקבוע קו-בסיס של הדרישות עבור ה Build המתוכנן. השמה בקו-בסיס כרוכה בד״כ בהכנסת תוצרים שונים (כגון : מסמך החזון , Use Cases , דרישות לא-פונקציונליות וכד׳) למסגרת של בקרת תצורה וסימונם באמצעות תגית מסויימת (Label) עבור האיטרציה המסויימת. באמצעות קו הבסיס קל לדעת עם אילו Use Cases יצאנו לדרך באיטרציה הנוכחית ואילו Use Cases יעברו שינוי באיטרציה הזו. במידה ושינוי שנדרש אושר (למשל ע״י ועדת שינויים) אזי קל לנהל את השינויים ואת ההשפעה של השינוי על תוצרים אחרים.
- ייסוד של ערוץ יחיד לבקרת השינוי.
  ישנה חשיבות כי ההחלטה לבצע את השינוי וניהול השפעתו תעבור דרך ערוץ יחיד ולא תתבדר דרך מספר גורמים רב , דבר אשר עשוי לפגוע באפקטיביות של יישום השינוי. בפרוייקטים קטנים בעל התפקיד שמקיים ערוץ זה הינו

בד״כ מנהל המוצר ובפרוייקטים גדולים תפקיד זה מורכב ממספר אנשים החברים בועדת שינויים[14].

- שימוש במערכת לבקרת שינויים.
באמצעות שימוש במערכת לבקרת שינויים ניתן לנהל את השינויים בצורה מיטבית. מרבית הכלים (כגון Clear Quest) תומכים בהזנת סכימה המגדירה את התהליך המתבצע לאורך חייו של השינוי. כאשר מוזנת בקשה לשינוי היא מתחילה להתגלגל בהתאם לתהליך הבקרה של שינויים בארגון כפי שהוזן בסכימה. במידה והכלי מקושר לכלי לניהול הדרישות ניתן גם לעדכן באופן אוטומטי את מטריצות העקיבות השונות. יתרונות נוספים של שימוש במערכת כזו הינן היכולות להפיק דו״חות של מדדי איכות הקשורים לשינויים במערכת. בדרך כלל מערכת לניהול שינויים מנהלת גם את התקלות הנמצאות במערכת המפותחת וזאת מכיוון שבקשה לתיקון כמוה כבקשה לשינוי.
- ניהול היראכי של השינוי.
מטרת תת-שלב זה הינה למנוע פספוס של החלת השינוי על כל האלמנטים המושפעים מן השינוי. בדרך כלל , סכימת ניהול השינוי מכילה שלב של ניתוח השפעת השינוי על כלל המערכת. בשלב זה נעזרים במטריצות העקיבות ומנסים להבין אילו שינויים נגררים בעקבות החלת השינוי הנוכחי. ניתוח השפעת השינוי תבוצע באופן היראכי על מנת לוודא כי לא הוזנחו אלמנטים נוספים.



איור18 – היראכיית אפקט השינוי

[14] CCB – Change Control Board

3.1.7. <u>קלטים</u>:

- ❖ מודל Use Case מלא (תוצרים סופיים משלב קודם)

4.1.7. <u>כלים וטכניקות</u>:

- ❖ שימוש בטבלת בדיקות לתיקוף.
- ❖ ביצוע סקר תיקוף פורמלי.
- ❖ שימוש במטריצות עקיבות –
  - Use Case מול אלמנטי דרישות (Use Cases, Scenarios, Features וכד').
  - Use Case מול אלמנטי בדיקות (Test Case וכד').
  - Use Case מול אלמנטי תכן ויישום.
- ❖ שימוש בכלי לניהול דרישות :
  - כגון Rational Requisite Pro
- ❖ שימוש בכלי לניהול שינויים:
  - כגון Rational Clear Quest

5.1.7. <u>איכות</u>:

- ❖ ביצוע הפעילויות השונות כחלק מהבטחת איכות המודל.

6.1.7. <u>סיכונים ושגיאות נפוצות:</u>

❖ טבלת בעיות אפשריות במודל ה Use Case (Anda and Sjoberg, 2002):

	שחקנים	**Use Cases**	**תזרים האירועים (צעדים)**	**וריאציות (הרחבות / טיפול בשגיאות)**	**יחסים בין Use Cases**	**טריגר ותנאים מקדימים ומאוחרים**
חוסרים	לא זוהו כל הישויות החיצוניות למערכת אשר מתקשרות עמה	לא כל הפונקציונליות הנדרשת מהמערכת מתוארת בעזרת Use Case. ישנם מספר מטרות של שחקנים שלא מיוצגות באמצעות שום Use Case.	קלט או פלט של מספר Use Cases לא מתואר כנדרש. אירועים הדרושים להבנת ה Use Case חסרים.	מספר וריאציות אשר עשויות להתרחש בעת נסיון להשגת מטרת ה Use Case אינן מתוארות.	אין ציון של יחס הכללה עבור Use Cases אשר בשימוש ע"י Use Cases אחרים. אין ציון של יחס הרחבה עבור Use Cases אשר מרחיבים Use Cases אחרים.	טריגר ו/או אחד מהתנאים הושמטו מהתיאור של ה Use Case.
מידע שגוי	תיאור שגוי של השחקנים. קשרים שגויים בין השחקן וה Use Case.	תיאור שגוי של ה Use Case.	תיאור שגוי של אירוע יחיד או מספר אירועים.	תיאור שגוי של אחת הוריאציות.	לא ישים	הנחות שגויות אשר גרמו לתנאים שגויים אשר אינם מתקיימים.
חוסר עקביות	תיאור השחקן בטבלת פרופיל שחקן איננה מתאימה לתיאורו בתבנית ה Use Case. תיאור השחקן איננו מתאים להתנהגותו	חוסר עקביות בין המטרה ב Use Case לבין תיאור ה Use Case.	חוסר תאימות בין אירועים לבין המטרה של ה Use Case בהם הם מופיעים.	חוסר עקביות בין הוריאציות לבין מטרת ה Use Case.	חוסר עקביות בין היחסים כפי שמתוארים בדיאגרמת ה Use Case לבין תיאורם בתבנית ה	התנאים אינם עקביים למטרת ה Use Case ו/או לאירועי ה Use Case.

	בUse Case.				Use Case.	
דו-משמעות	תיאור כללי מדי של השחקנים. תיאור דו-משמעי של שחקן.	שם ה Use Case איננו משקף את מטרתו.	תיאור דו-משמעי של אירוע (יתכן עקב מחסור בפרטים).	תיאור דו-משמעי אשר מוביל לתיאור וריאציה מסוימת.	לא ישים	תיאור דו-משמעי של טריגר / תנאים.
מידע מיותר	תיאור שחקנים אשר אינם מביאים ערך למערכת.	תיאור Use Cases מחוץ לסקופ של המערכת המתוארת או שכוללים פונקציונליות אשר כבר נכללה ב Use Case אחר.	צעדים מיותרים. עודף פרטים בתיאור הצעד.	תיאור וריאציות אשר מחוץ לסקופ של אותו Use Case.	לא ישים	תיאור מיותר של טריגר / תנאים.
תוצאות	הפונקציונליות המבוקשת איננה זמינה עבור מספר משתמשים. ממשק למערכות אחרות חסר.	פונקציונליות מבוקשת חסרה.	יותר מדי אילוצים או אילוצים שגויים על התכן. מטרת השחקן איננה מושגת.	טיפול בשגיאות נזנח. הפרדה בין פונקציונליות שגויה.	אי הבנות בין בעלי עניין שונים. תכן לא יעיל.	קושי לבדיקת המערכת. קושי לנווט בין Use Cases שונים.

טבלה27

❖ שגיאות נפוצות עבור פעילויות הניהול:

	מטריצת עקיבות	**ניהול הדרישות**	**ניהול השינויים**
חוסרים	קשר תלות בין Use Case לאלמנט אחר איננו מופיע במטריצה.	אחד ה Use Cases במודל איננו מנוהל באמצעות הכלי לניהול הדרישות.	בוצע שינוי ב Use Case ללא בקרה על השינוי כמתחייב מתהליך ניהול השינויים.
מידע שגוי	קשר תלות בין Use Case לאלמנט אחר מופיע בכיוון ההפוך במטריצה.	Use Case עם גירסה לא עדכנית מנוהל בכלי לניהול דרישות עם תגית שגויה.	בוצע שינוי ב Use Case אשר איננו תואם את השינוי שאושר לביצוע ע"י ועדת השינויים.
מידע מיותר	קשר תלות שאיננו קיים בין Use Case לאלמנט אחר מופיע במטריצה.	Use Case שאיננו קיים (נמחק, התמזג וכד') עדיין מנוהל כיחידה בכלי לניהול הדרישות.	בוצעו שינויים לא הכרחיים ב Use Case אשר מוסיפים מידע אשר קיים כבר ב Use Case.

טבלה28

❖ שגיאות נפוצות במהלך ניהול המתודולוגיה (Kulak and Guiney ,2000):

השגיאה	**מה עשוי להגרם / הנחיות**
ניסיון לכפות ביצוע של איטרציות במקביל.	האיטרציות ניתנות לביצוע באופן מקבילי ואף מתוכננות לכך. אולם, ישנם מנהלים שמאלצים זאת ולעיתים מקדימים ביצוע של האיטרציה הבאה על מנת לעמוד בלוח הזמנים. את האיטרציה הבאה יש להתחיל כאשר באופן טבעי האיטרציה הקודמת הבשילה וניתן כבר לצלול לרמה הבאה של ה Use Cases. הבשלה של איטרציה מתבטאת בסיום זיהוי של ה Use Cases העיקריים ברמה הנוכחית.
חוסר איזון של נסיון בחלוקת התפקידים.	לעיתים ישנם מספר מנתחי מערכות / מהנדסי דרישות בעלי נסיון אשר מכירים את התחום העסקי של המערכת וישנם אחרים בעלי פחות נסיון. יש ליצור איזון בחלוקה ביניהם על מנת להמנע ממצב בו חלק מהמערכת מנותח ברמה הגבוהה ביותר וחלק אחר מוזנח.
איחוד של Use Cases לחבילות מאוחר מדיי.	איגוד לחבילות הינו כלי להקטנת הסיבוכיות. יש להשתמש בכלי זה לעיתים תכופות ומוקדם ככל הניתן.
שימוש בחבילות על מנת להסתיר חלקים בלתי ברורים של המערכת.	האיגוד לחבילות נועד להסתיר מורכבות ממנה נרצה להמנע אך לא נועד להסתיר חלקים במערכת שאיננו מבינים. איגוד לחבילות של חלקים אותם איננו מבינים על מנת לדחות זאת להמשך מגדיל את הסיכונים באופן משמעותי ועלול לגרום לתגובת שרשרת גדולה של שינויים בהמשך.

טבלה29

2.7. תוצר נדרש: מודל Use Case משולב

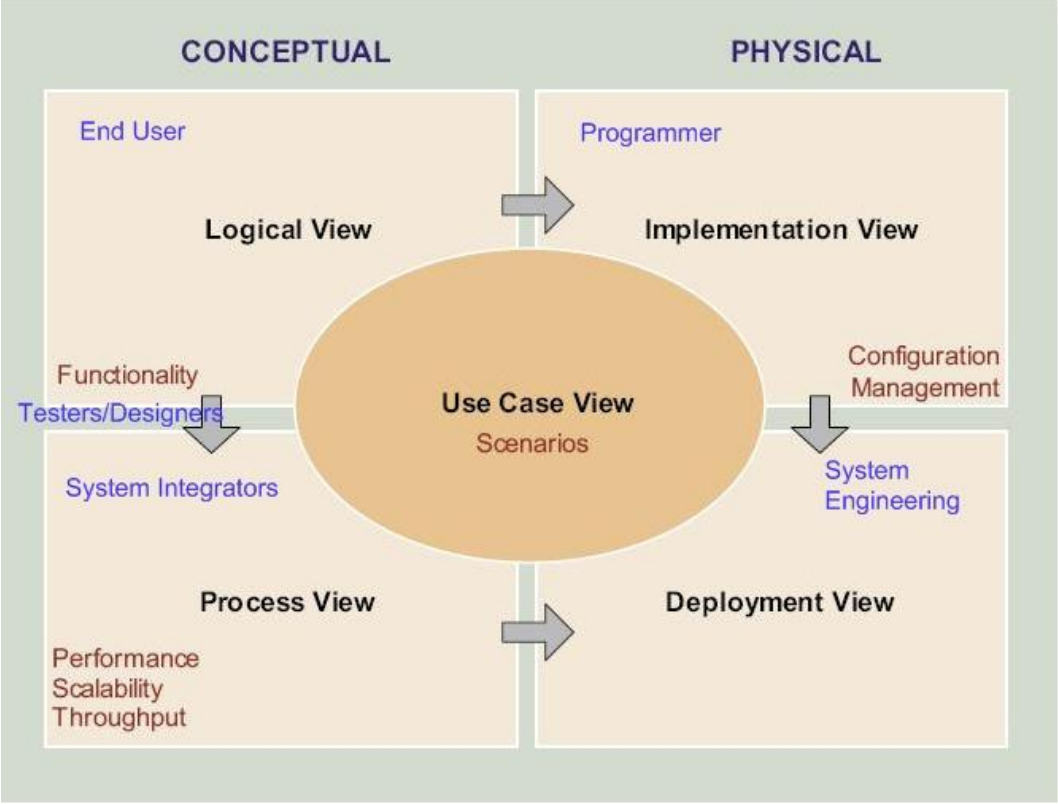
מודל Use Case משולב הינו מודל Use Case אשר כולל את כל המרכיבים והתרשימים הנדרשים על מנת להשתלב כ-View נוסף של המערכת.

1.2.7. המטרה:

שילוב מודל ה Use Case בארכיטקטורת המערכת.

2.2.7. פעילויות:

- שילוב המודל בארכיטקטורת המערכת (Heeseok and Keunhyuk, 2002) ארכיטקטורת 4+1: בפעילות זו נתייחס למודל ה Use Case כאל View אחד של המערכת המקשר בין 4 Views נוספים כפי שהוצע ע"י Philippe Kruchten ב 1995



- Logical View

מייצג את פונקציונליות המערכת. בעזרת הפשטה זו נייצג את המבנה הלוגי של המערכת במונחים של תת-מערכות ומחלקות. אלו היישויות אשר מוסרות את פונקציונליות המערכת למשתמש.

- Implementation View

מייצג את הביטים ושאר הרכיבים הרלוונטים ליישום המערכת. כולל קוד מקור, ספריות , אובייקטי מחלקות וכו'.

- Process View

מייצג את התהליכים הרצים במהלך הפעלת המערכת. זה חשוב ביותר כאשר המערכת מורכבת ממספר תהליכים הפועלים במקביל. View זה מאפשר לאתר בעיות פוטנציאליות של פעילות מקבילית כגון deadlocks.

- Deployment View

מייצג את הקצאת אלמנטי הישום לסביבה בה המערכת פועלת. View זה מתאר את האינטרקציה של המערכת אל מול רכיבים כגון מערכת ההפעלה והפלטפורמה ע"ג המערכת רצה.

❖ ארכיטקטורת המערכת חייבת לתמוך בדרישות הפונקציונליות של המערכת המפותחת ולשם כך על ארכיטקט המערכת להעזר ב Use Cases העיקריים המתארים את תמצית המערכת (Malan, and Bredemeyer ,2001). תהליך זה עשוי להתרחש גם בכיוון השני (Leffingwell and Widrig ,2005).

- בעזרת שילוב המודל בארכיטקטורה הנ"ל נזהה את ה Use Cases המרכזיים אשר מהווים את ה"דבק" בין ה Views השונים של המערכת. הפעילות מסייעת בהבנה של הקשרים בין הUse Cases לבין יתר התוצרים המנוהלים במהלך פיתוח המערכת. קשרים אלו מחייבים שימוש במטריצת קשירויות.

במהלך פעילות זו נשלב את המודל בכל ה Viewים השונים. השילוב ייעשה תוך כדי יצירת תרשימי UML המתאימים לכל View.

3.2.7. <u>קלטים</u>:

❖ מודל Use Case שלם ומתוקף.

4.2.7. כלים וטכניקות :

- ❖ שימוש במטריצת קשירויות לזיהוי הקשרים בין ה Use Cases לאלמנטים אחרים במערכת.
- ❖ שימוש בתרשימים תקניים של UML על מנת לתאר את המודל בכל הViewים השונים.

5.2.7. איכות :

- ❖ יש לוודא כי כל תרשימי ה UML אכן תקניים בהתאם להנחיות UML.

6.2.7. סיכונים ושגיאות נפוצות :

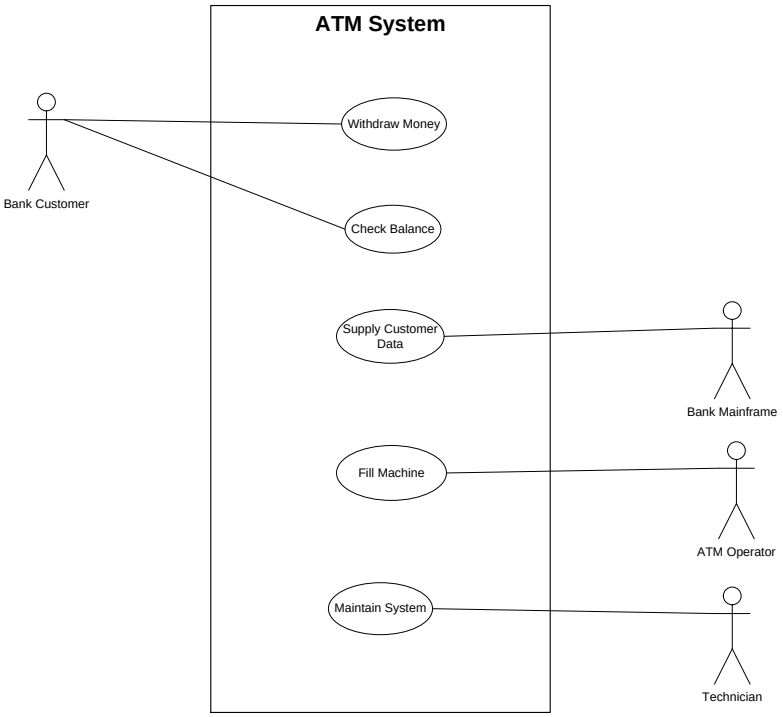
השגיאה	**מה עשוי להגרם / הנחיות**
יצירת מודל Use Cases לא התחשבות בארכיטקטורת המערכת	סתירות בין ייצוג הדרישות ע״י המודל לבין הדרישות בהתאם לאריכטקטורת המערכת. ישנה השפעה הדדית בין ארכיטקטורת המערכת לבין דרישות המערכת.
שילוב אלמנטים מעולם המימוש בתרשימים המתארים את עולם התכן	יגרום להקשחה של אופן היישום.
שחקנים המופיעים במודל ה Use Case אינם מופיעים בLogical View.	חוסר עקיבות. חייב להיות ייצוג כלשהו לכל שחקן אשר זוהה במודל ה Use Case גם בשלב התכן. בד״כ הייצוג יהיה בצורת ממשק או מחלקת גבול.

טבלה30

7.2.7. דוגמא לשילוב המודל בארכיטקטורת המערכת עבור מערכת ATM (Heeseok and Keunhyuk, 2002):

❖ Use Case View:

יכיל תרשים Use Case מערכתי כייצוג ואת יתר המודל באופן מפורט (כולל כל ה Use Cases המתוקפים).



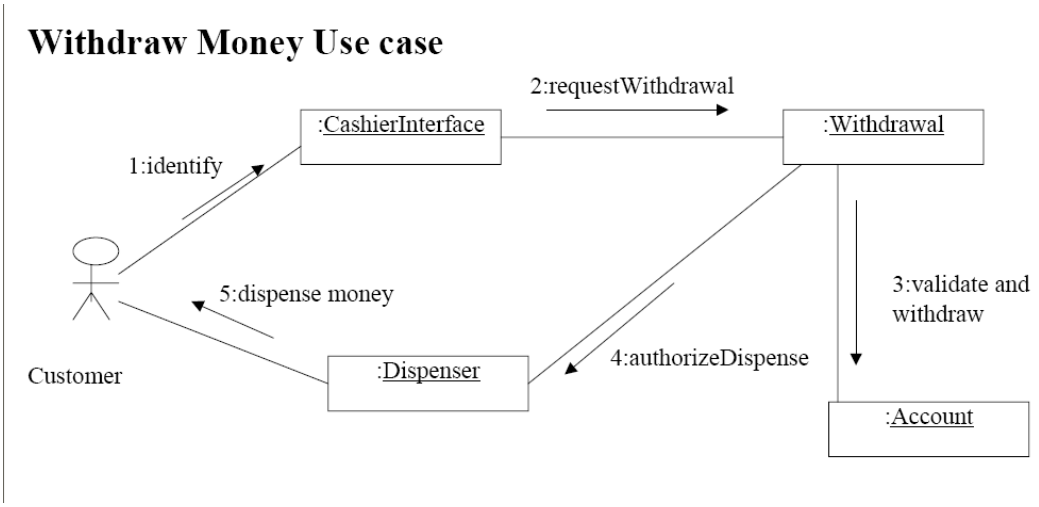
איור20

מודל ה Use Case על שלל תוצריו מהווה את ה-View המקשר בין יתר ארבעת ה-Views של ארכיטקטורת המערכת.

08/01/09

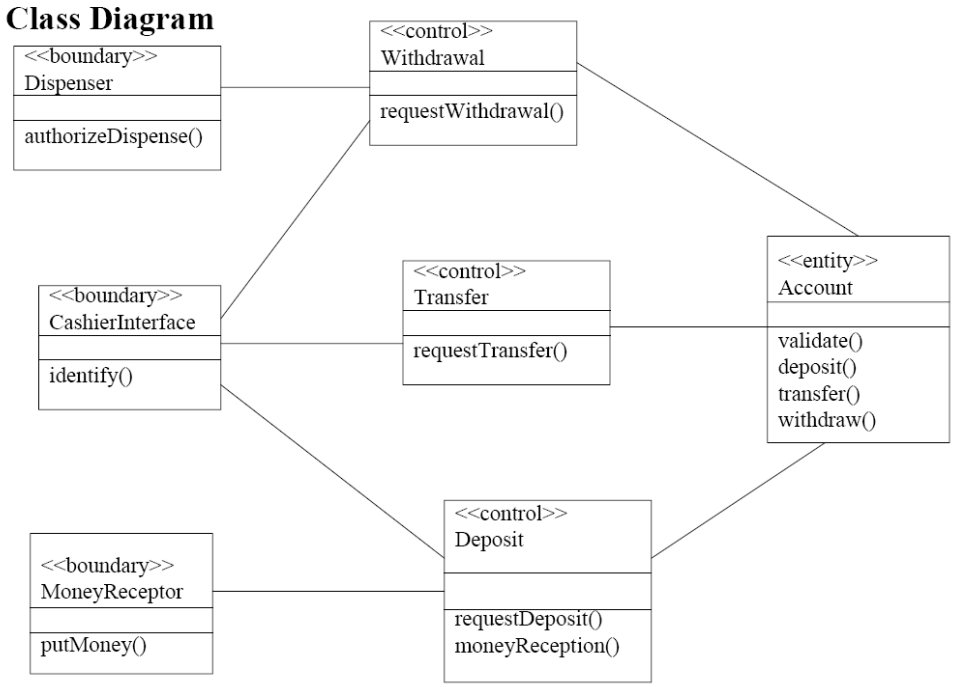
❖ Logical View :

יכיל תרשימים המתארים את תכן המערכת (כגון תרשימי Class , Interaction,State). יתאר את תת המערכות והממשקים.



איור21

התרשים הנ"ל מתאר בצורה גרפית Use Case של "משיכת כסף" מהמערכת המתבצעת ע"י הלקוח בחמישה שלבים עיקריים.

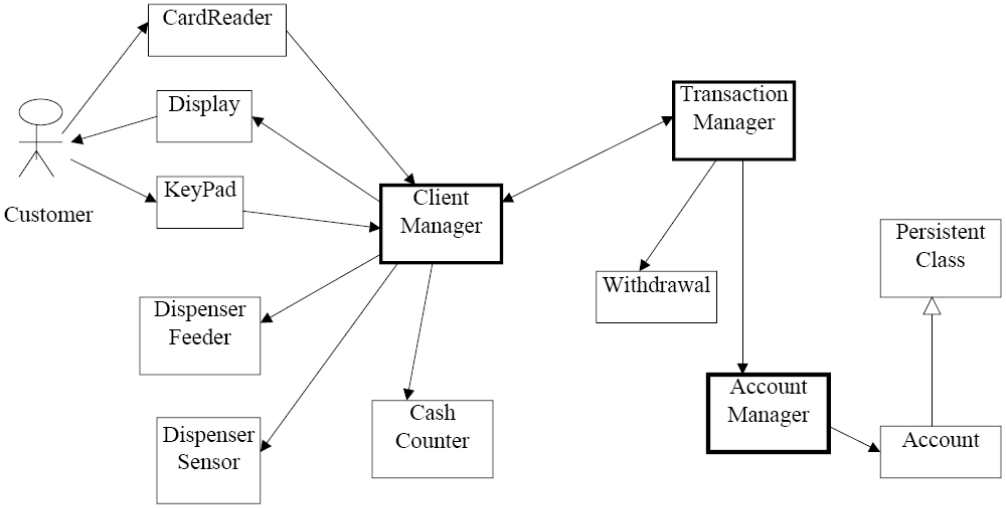


איור22

08/01/09

התרשים הנ״ל הינו דיאגרמת מחלקות ברמת על של המערכת. עבור כל מחלקה אנו מציינים את סוג המחלקה בתוך סטריאוטייפ (לדוגמה : מחלקת Dispenser הינה מחלקת גבול <<boundary>>). בחלקה התחתון של כל מחלקה אנו מציינים את הפעולות העיקריות אותן ניתן לבצע במסגרת אותה מחלקה (לדוגמה : ()PutMoney זו פונקציה השייכת למחלקת MoneyReceptor).

(Refined) Class diagram providing a view of the classes involved in withdraw Money use case (design model)

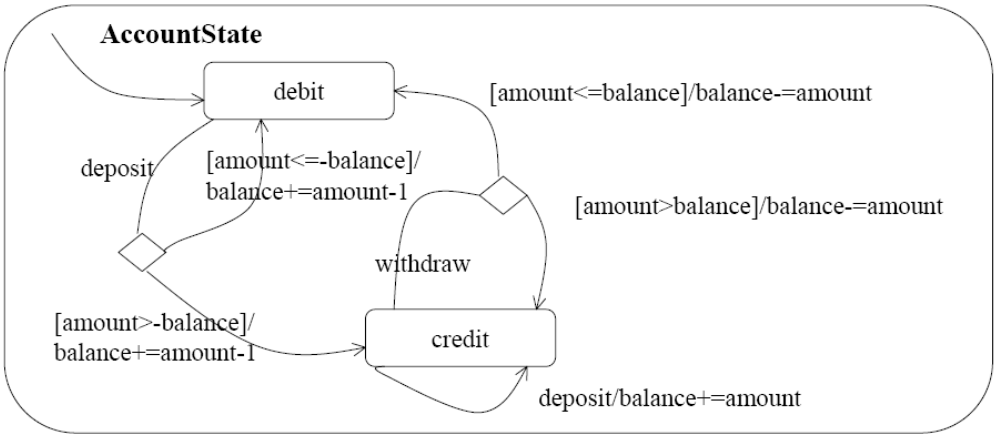


איור23

עבור כל Use Case ניתן לייצר תרשים מחלקות משלו. בתרשים הנ״ל בוצע עידון של דיאגרמת המחלקות הראשית וכעת מוצגות מחלקות נוספות המעורבות בתהליך ״משיכת מזומנים״ מהמערכת ע״י הלקוח.

08/01/09

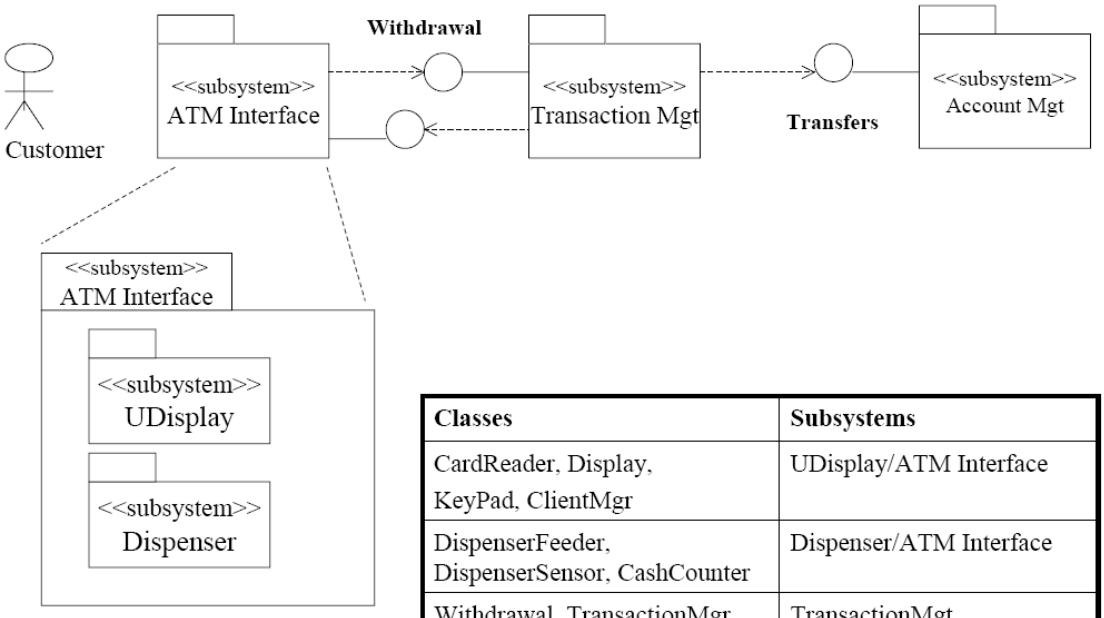
Statechart Modeling Dynamic Behavior of Account Class



איור24

התרשים הנ״ל זהו תרשים מצבים המתאר התנהגות דינמית של תהליכים המתרחשים במחלקת AccountState. התהליך מתאר את עדכון יתרת הלקוח לאחר ביצוע משיכת המזומנים.

Static Structure Diagram



Classes	Subsystems
CardReader, Display, KeyPad, ClientMgr	UDisplay/ATM Interface
DispenserFeeder, DispenserSensor, CashCounter	Dispenser/ATM Interface
Withdrawal, TransactionMgr	TransactionMgt
Account, PersistentClass, AccountMgr	AccountMgt

איור25

התרשים הנ״ל הינו תרשים סטטי המתאר ממשקים בין תתי מערכות ומהן המחלקות העיקריות בכל תת מערכת.

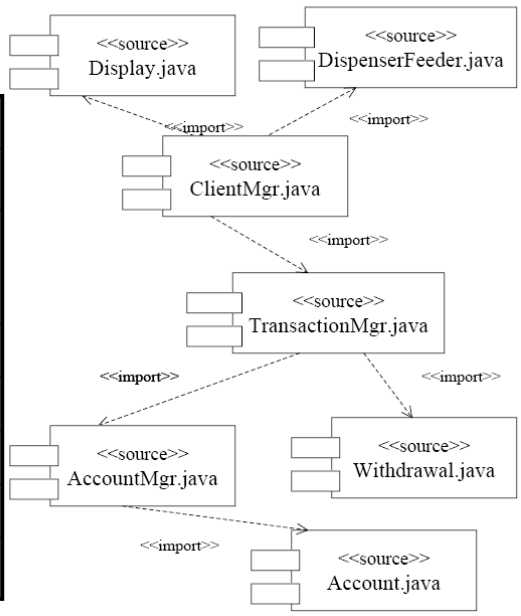
❖ Implementation View

יכיל את תוצרי המימוש ותרשימים מתאימים (כגון קוד מקור , קבצי הרצה וכד׳)

Implementation View

-Source Components

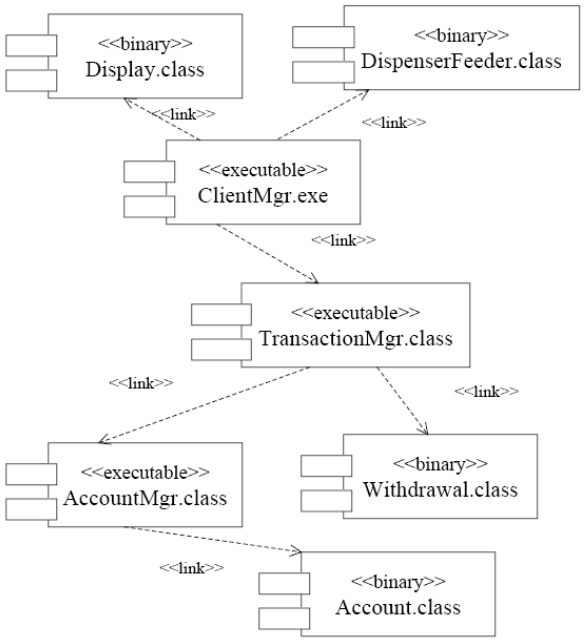
Classes	Source Components
CardReader, Display, KeyPad	Display.java
DispenserFeeder , DispenserSensor , CashCounter	DispenserFeeder.java
ClientMgr	ClientMgr.java
TransactionMgr	TransactionMgr.java
AccountMgr	AccountMgr.java
Withdrawal	Withdrawal.java
Account, Persistent class	Account.java



איור26

התרשים הנ״ל מתאר כבר את המחלקות הממומשות בעבור המערכת. ישנו מיפוי בין כל מחלקה לשם קובץ המקור.

-Executable Release



איור27

08/01/09

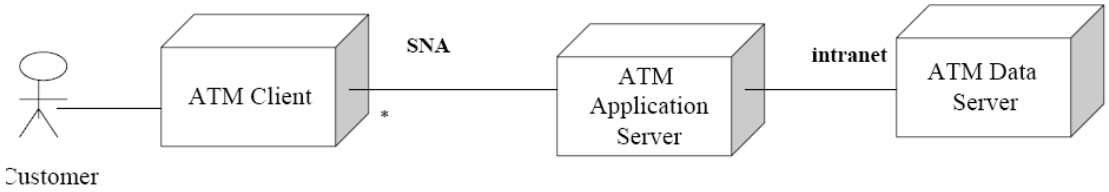
התרשים הנ״ל מתאר מימוש ברמה של קבצי הרצה. מתוארות התלויות בין הבינאריים השונים של המערכת.

❖ Process & Deployment View

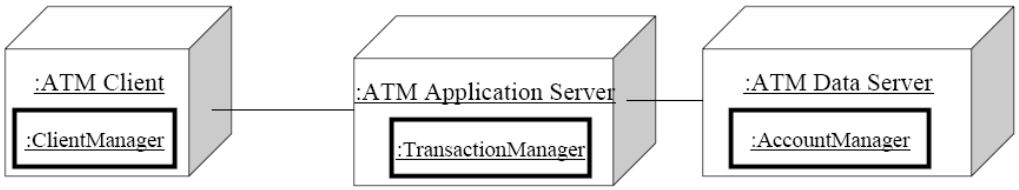
מייצג את הקצאת אלמנטי הישום לסביבה בה המערכת פועלת

Deployment View

- Deployment Diagram



- Deployment of Active Objects



איור28

התרשים הנ״ל מתאר את תהליך הפריסה של המערכת. כיצד המערכת מופצת ללקוח ומהם פרוטוקולי התקשורת שבהן המידע מועבר מקצה (לקוח) לקצה (מחשב השרת).

סיכום:

אחת ממטרות המשנה של שימוש במתודולוגיה ליצירת הUse Cases הינה איכות. בניית מודל Use Cases בצורה שיטתית כאשר בכל שלב ישנם מדדי איכות לתוצרים השונים. בתום בניית המודל אנו מגיעים לשלב זה ובו אנו מתקפים את המודל. מעבר לתיקוף המודל יש לנהל את התוצרים השונים לאורך כל שלבי יישום המתודולוגיה. ניהול המודל כולל ניהול תצורה וניהול שינויים באמצעות תהליך מובנה ושימוש בכלים שונים. לבסוף ,המודל משולב בארכיטקטורה הכללית של המערכת. שילוב וניהול מודל ה Use Case מותאם בכל ארגון בהתאם לנוהלי הבטחת איכות התוכנה בארגון.

❖ בעלי תפקידים:

- ארכיטקט מערכת – שילוב מודל ה Use Case בארכיטקטורה הכללית של המערכת.
- מנהל תצורה – ניהול תצורה וקווי בסיס של ה Use Cases תוך שימוש בכלי לניהול תצורה (כגוןRational Clear Case).
- וועדת שינויים – ניהול ואישור השינויים ב Use Cases תוך שימוש בכלי לבקרת שינויים (כגון Rational Clear Quest).
- מנהל פרויקט – ניהול כללי של מודל ה Use Cases תוך וידוא שמירה על נהלי הבטחת איכות התוכנה (יצירת מטריצות עקיבות, ניהול תצורה וכו')
- ממונה הבטחת איכות תוכנה – וידוא ביצוע כלל הפעילויות בהתאם לנהלים ועקרונות הנדסת תוכנה.

❖ רשימת תוצרים:

להלן רשימה של התוצרים השונים המופקים בשלב זה:

התוצר	שלם/חלקי
מודל Use Cases מתוקף	שלם
מודל Use Cases משולב	שלם

טבלה31

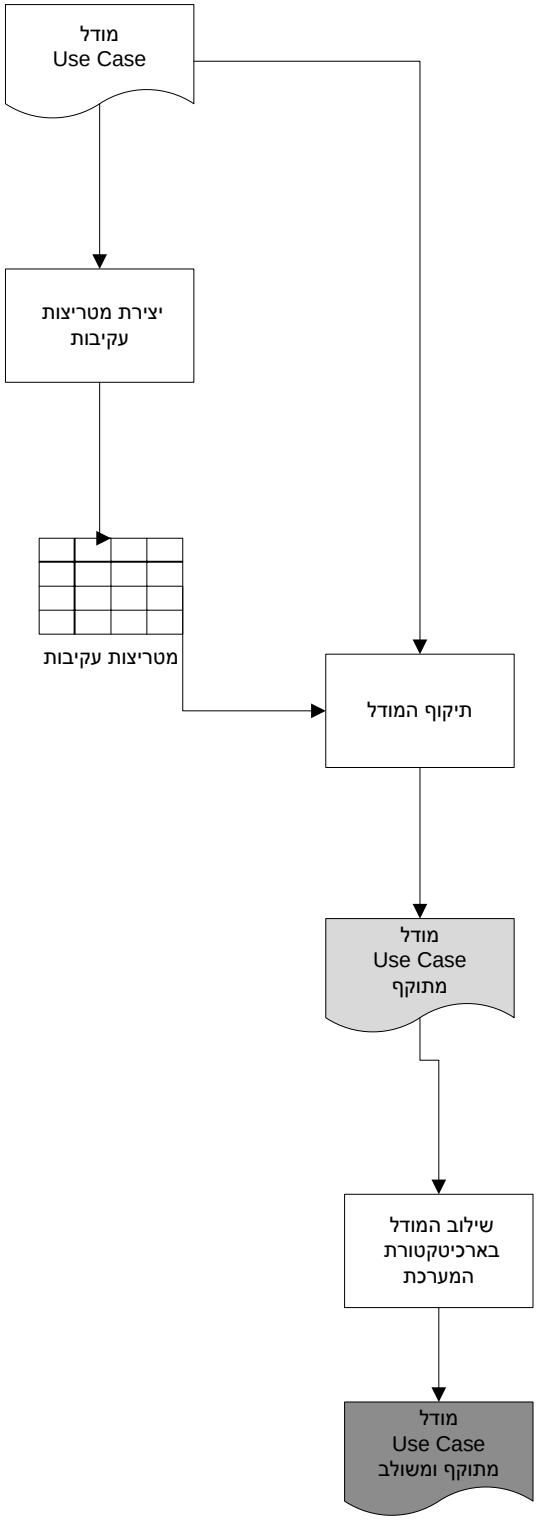
❖ סיכום הסיכונים העיקריים בשלבים השונים :

שלב	מטרה עיקרית	סיכונים עיקריים
תיחום גבולות המערכת	קביעת ה SCOPE של המערכת	קביעת גבולות מערכת שגויים
זיהוי שחקנים ומטרות	זיהוי בעלי העניין, השחקנים , המטרות והאינטרסים שלהם מהמערכת.	פספוס של מטרות ושחקנים
גיבוש Use Case שלדי	זיהוי Use Cases עיקריים במערכת וכתיבה תמציתית שלהם.	פספוס Use Cases. הבנה שגויה של הסקופ. שיוך Use Case לרמה לא מתאימה (צריך להתבצע באיטרציה אחרת)
הרחבת ה Use Case	כתיבה שלמה של ה Use Cases כולל טיפול בהסתעפויות. איגוד לחבילות סופיות.	חוסר שלמות – המודל איננו משקף את דרישות המערכת. המודל איננו ברור וקשה לתחזוקה.
הבטחת איכות המודל	הבטחת איכות למודל. ניהול הדרישות והשינויים. שילוב המודל בארכיטקטורת המערכת.	אובדן שליטה בניהול דרישות המערכת. פספוס השפעות שינויים. איכות מודל ירודה.

טבלה32

08/01/09

להלן תרשים מסכם לשלב V – הבטחת איכות המודל :



8. **סיכום** :

<u>בפרק הראשון</u> – "מבוא" הוצג הנושא של מודל ה Use Cases וניתנו מספר הגדרות עבור Use Case. הוצגה הבעיה שעבודה זו באה לתת מענה עבורה: מחסור במתודולוגיה סדורה עבור Use Cases. בפרק זה הוגדרו מספר מונחים לצורך יצירת טרמינולוגיה להמשך.

<u>בפרק השני</u> – "מרכיבי ה Use Cases" הוצגו והוגדרו כל מרכיבי ה Use Case. הוצגו תבניות שונות לתיאור Use Cases ולבסוף הוצג נושא תרשימי ה Use Cases. בפרק זה הגדרנו את כל רכיביו הבודדים של ה Use Case והצגנו דרכים שונות לתיאורו ובכך נוצר בסיס משותף עם הקורא לקראת הצגת המתודולוגיה עצמה.

<u>הפרק השלישי</u> – "מתודולוגיה לבניית Use Cases" הינו הפרק העיקרי בעבודה זו. בפרק זה הוצג מבנה המתודולוגיה ומרכיביה:

1. מטרות.
2. קלטים.
3. פעילויות.
4. כלים וטכניקות.
5. תוצרים.
6. איכות.
7. ארגון מבני.
8. בעלי תפקידים.
9. שגיאות נפוצות וסיכונים.

<u>לאחר מכן הוצגו השלבים השונים של המתודולוגיה</u>:

I. תיחום גבולות המערכת – השלב הראשוני בו אנו מבצעים Scoping, מארגנים את המידע על המערכת וקובעים מה נכלל בפיתוחה ומה נמצא מחוץ לגבולות המערכת.

II. זיהוי שחקנים ומטרות –בשלב זה אנו הולכים צעד אחד קדימה ומזהים את השחקנים אשר יש להם אינטראקציה עם המערכת. זיהוי השחקנים מבוצע באופן שיטתי כאשר מטרות השחקנים הינם הוקטור המוביל את התהליך.

III. גיבוש Use Case שלדי – בשלב זה אנו כבר מתחילים ליצור את ה Use Cases. הבנייה מתבצעת באופן שיטתי והדרגתי כך שבשלב זה אנו יוצרים שלד Use Case

בלבד עבור רמת ה Use Case בה האיטרציה הנוכחית עוסקת. בתום שלב זה בידינו חבילות של Use Cases בתצורה שלדית המכילה תיאור כללי של תרחיש ההצלחה.

IV. הרחבת ה Use Case – בשלב זה אנו משלימים את שלדי ה Use Cases, מפרטים את התרחישים השונים כולל הסתעפויות ותרחישי כשלון. גם שלב זה מתבצע באופן איטרטיבי. בתום שלב זה אנו יוצרים אוסף של חבילות Use Case שלמות המהווה את מודל ה Use Case המלא עבור המערכת המפותחת.

V. ניהול ה Use Cases והבטחת איכות – בפרק זה אנו מפרטים את פעילויות הניהול והבטחת האיכות אשר מבוצעות לאורך כל שלבי הפיתוח. בכל שלב אנו מבצעים פעילויות וסעיף האיכות נותן לנו כלים לאמת את התוצרים. בפרק זה מוצגות הפעילויות שאינן ספציפיות לשלב מסויים אלא רוחביות במתודולוגיה המוצעת.

נושא ה Use Case הינו נושא שאיננו פורמלי בבסיסו. על כן קיימות הגדרות שונות ל Use Case וזוויות ראייה רבות לאופן השימוש בו. המתודולוגיה מנסה להביא את העקרונות הכלליים והטובים מכל העולמות ולאפשר התאמתה למקרים שונים. המתודולוגיה שהוצגה בעבודה זו הינה מתודולוגיה מעשית הניתנת ליישום כמעט בכל סוג של מערכת. המתודולוגיה נבנתה ע"פ העקרונות המקובלים בספרות ונסמכת על מקורות רבים כפי שמצויין ברשימת המקורות. יש לציין כי עקרונות רבים המופיעים בספריו ומאמריו של Alistair Cockburn הם שהיוו את הבסיס ליצירת המתודולוגיה. במהלך הצגת המתודולוגיה הוצגה מערכת לדוגמא לצורך מידול בשיטת ה Use Cases. המערכת שתוארה הינה מערכת של כספומט (ATM) המאפשרת לבצע מספר פעולות כגון משיכת מזומנים. מרבית הדוגמאות לפעילויות, תוצרים, תרשימים ועוד הובאו בהקשר של מערכת ה ATM.

המתודולוגיה מלווה בפעילויות של הבטחת איכות אשר יסייעו לקורא ליישם את העקרונות ואף לבצע בקרה ואימות של הפעילויות השונות. בנוסף ישנה התייחסות לאורך כל העבודה לשגיאות נפוצות וסיכונים על מנת שהקורא שיישם את המתודולוגיה יוכל ללמוד מטעויות אלו ולהשתדל להמנע מהן.

08/01/09

<u>מגבלות המתודולוגיה והצעות לכיווני מחקר נוספים</u> –

במסגרת העבודה הנ״ל לא ניתן היה להציג מידול מלא של מערכת אמיתית באמצעות המתודולוגיה. הדוגמא שניתנה (ATM) הינה מוגבלת ומייצגת מערכת פשוטה ולא מורכבת כמערכות רבות ב״חיים האמיתיים״. מידול של מערכת אמיתית וגדולה הינו תהליך ארוך וכולל מעורבות של אנשי פיתוח רבים ותוצרים רבים העשויים להתפרש על פני מאות עמודים. המתודולוגיה איננה מציעה ליצור תחליף למפרט דרישות תוכנה פורמלי (SRS) אשר בניגוד למודל ה Use Case מכיל גם אילוצים ודרישות לא-פונקציונליות. במסגרת עבודה זו לא מוצגות הדרכים לשילוב המתודולוגיה עם מפרט הדרישות. מצד אחד Use Cases אינם יכולים להכיל את כלל הדרישות ומצד שני אנו רוצים להמנע מכפילות בתחזוקת הדרישות ועל כן שילוב המתודולוגיה עם מפרט הדרישות הפורמלי הינו נושא שיש להעמיק בו ולחקור. מודל ה Use Case איננו מתאים לכל סוג פרוייקט. המודל יהיה פחות אפקטיבי בפרוייקטים עתירי בסיסי-נתונים , מערכות משובצות מחשב , מערכות מרובות רכיבי חומרה ואפליקציות עתירות חישובים. בפרוייקטים אלו האינטרקציה בין המשתמש למערכת הינה שולית ונסיון לכפות את מודל ה Use Case כמודל עיקרי דינו להכשל (Wiegers, 2006).

במסגרת עבודה זו אין התייחסות להמשך הפיתוח מתוצרי ה Use Cases ליתר התוצרים ועד לקוד תוכנה כפי שמוצע במתודולוגיות אחרות כגון ICONIX. נושא זה הינו נושא מעניין בפני עצמו ויכול להוות נושא לעבודת המשך.

כיוון מחקר נוסף אשר עשוי להיות מרתק הינו פיתוח שיטה לייצוג פורמאלי של המתודולוגיה ויצירת מדדי איכות כמותיים וע״י כך הרחבת הבסיס ההנדסי והמדעי של המתודולוגיה המוצעת.

9. נספחים :

1.9. הגדרות Use Case :

להלן מספר הגדרות שונות עבור Use Case :

- ע"פ Jacobson , 1999 :

"A use case specifies a sequence of actions, including variants that the system can perform and that yield an observable result of value to a particular actor."

- ע"פ Cockburn , 2000 :

"A use case captures a contract between the stakeholders of a system about its behavior. The use case describes the system's behavior under various conditions as the system responds to a request from one of the stakeholders, called the prinary actor."

- ע"פ Bittner & Spence , 2002 :

"A use case describes how an actor uses a system to achieve a goal and what the system does for the actor to achieve that goal. It tells the story of how the system and its actors collaborate to deliver something of value for at least one of the actors."

- ע"פ Övergaard ,2004 :

"A use case models a complete usage, including variants, of a system. It provides a business value to one of the stakeholders of the system. A concrete use case is always initiated by an actor."

ישנן הגדרות רבות נוספות ל Use Case , כאשר בכל הגדרה ישנו דגש על אספקט אחר. מרבית ההגדרות מציינות כי Use Case הינו כלי לתיאור התנהגות המערכת כאשר מתרחשת אינטראקציה בין "שחקן" לבין המערכת. ההגדרות גם מתייחסות לכך כי ל"שחקן" או "בעל העניין" ישנה מטרה או אינטרס מסויים ועל מנת להשיגו הוא מבצע פעולות מסויימות ובדרך כלל מתניע את התהליך כולו.

Mills (2002) טוען כי הגדרות רבות אינן חד משמעיות, סותרות זו את זו ומסתירות את המטרה העיקרית לשמה אנו יוצרים את ה Use Cases. הוא מציע להצמד להגדרת UML (OMG, 1999) :

"The specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system."

עבודה מסכמת לתואר שני

08/01/09

מעבר ליתרון של הגדרה זו על פני האחרות הנעוץ בכך כי הגדרה זו הינה מתוקננת ישנו יתרון נוסף שהגדרה זו מבטאת את ה Use Case במונחים של מפרט (specification) אותו ניתן לבדוק באמצעים של הבטחת איכות.

2.9. מונחים בנושאי Use Cases:

להלן תמצית הגדרות של מספר מונחים אשר נתייחס אליהם בהמשך העבודה בהרחבה.

- *שחקן* – מישהו או משהו עם התנהגות.
- *בעל עניין* – מישהו או משהו בעל אינטרס בהתנהגות המערכת הנדונה (SuD).
- *שחקן ראשי* – בעל העניין אשר מתחיל אינטראקציה עם המערכת הנדונה על מנת להשיג מטרה.
- *Use Case* – חוזה של התנהגות המערכת הנדונה.
- *Scope* – מזהה של המערכת בה אנו דנים.
- *תנאים מקדימים* (Preconditions) – מה חייב להתקיים לפני שאנו "מריצים" את ה Use Case.
- *ערבויות / תנאים מאוחרים* (guarantees) – מה חייב להתקיים לאחר שאנו "מריצים" את ה Use Case.
- *תרחיש ראשי להצלחה* (Main Success Scenario) – המקרה בו שום דבר איננו משתבש.
- *הרחבות* (Extensions) – מה יכול לקרות אחרת במהלך התרחיש.

3.9. מה זה Use Case:

Use Cases הם למעשה תיאור של רצף אירועים אשר יחדיו מביאים את המערכת לעשות משהו מועיל. כאשר הדרישות מרוכזות כערימה אחת זה קשה מאוד להבין מה רצו באמת כותבי הדרישות שהמערכת תבצע. Use Cases מקטינים את גורמי אי הוודאות ומסייעים בהבנה עמוקה יותר של המערכת. הם מתארים את התנהגות המערכת תחת תנאים שונים בזמן שהמערכת מגיבה לבקשה של אחד מ*בעלי העניין* של המערכת אשר נקרא *השחקן הראשי*.

לדוגמא, אם ה Use Case נועד לצורך תפיסת הדרישות של מודול מסויים בתוכנה אז המערכת הנדונה (SuD[15]) הינה תוכנית המחשב כולה ובעלי העניין עשויים להיות אלו שמשתמשים בתוכנה, החברה שקנתה את התוכנה ותוכניות מחשב אחרות. השחקן הראשי במקרה זה יהיה משתמש הקצה אשר יושב מול המסך ומפעיל את התוכנה ואולי גם מערכת מחשוב נוספת המפעילה את התוכנה.

Use Case צריך להיות קריא וברור. Use Case במהותו איננו גרפי אלא אוסף של צעדים אשר מתוארים בעזרת משפטים בתצורה אחידה. בכל צעד מתוארת פעולה אשר בעזרתה

---

[15] System Under Discussion

השחקן משיג תוצאה או מעביר אינפורמציה לשחקן אחר. ללמוד לקרוא Use Case זו פעולה שלא צריכה לארוך יותר מדקות ספורות. אולם ללמוד לכתוב Use Case בצורה נכונה זו כבר משימה הרבה יותר קשה. על מנת להצליח לכתוב Use Case טוב ,ישנם שלושה רעיונות בסיסיים שיש לזכור לאורך כל הדרך :

- Scope - מהי באמת המערכת בה אנו דנים? (SuD)
- שחקן ראשי – מיהו בעל המטרה להשגה (Goal) ?
- רמה (Level) – כמה נמוכה או גבוהה הרמה של המטרה ?

4.9. מה זה לא Use Case :

ישנן טעויות נפוצות רבות בעת כתיבת Use Cases וחשוב להיות מודע להן בבואנו לכתוב Use Cases. Use Cases הם לא פונקציות והם לא Features (Berard, 1998) למרות שדיאגרמת Use Cases לעיתים מזכירה תרשימי DFD[16] , לא ניתן לפרק בועות Use Cases כפי שמפרקים בועות בתרשימי ה DFD. בדיאגרמת Use Case ישנו קשר חזק בין הבועות השונות ולכן פירוק המערכת להמון Use Cases קטנים יגרום להסתרה של מטרת המערכת. אנו עשויים לסיים את המידול כאשר בידנו חתיכות רבות של התנהגויות שונות ובלתי ברורות של המערכת וכתוצאה מכך לא נוכל לומר מה על המערכת באמת לבצע.

Use Case מתאר התנהגות וע"י כך חושף את הדרישות מהמערכת. בבואנו לתאר Use Cases עלינו להמנע מהפרה של עקרון ההסתרה (Information Hiding). מפתחים רבים נוטים להתפתות ולצלול מעבר לממשקים הציבוריים ולתאר מבנים פרטיים שהינם חלק מהארכיטקטורה והתכן (Biddle , Noble and Tempero, 2001).

---

[16] Data Flow Diagram

5.9. מרכיבי ה - Use Case :

1.5.9. תבניות Use Case

מכיוון ש Use Case איננו כלי סטנדרטי התפתחו במרוצת השנים תצורות שונות ומגוונות לתיאור Use Case . נתאר את העיקריות שבהן :

❖ **תבנית מלאה ("Fully Dressed")**

להלן התבנית המוצעת ע"י Alistair Cockburn (Cockburn, 2000) :

Use Case #__ Use Case Name:

Primary Actor:

Scope:

Level:

Stakeholders & Interests:

Precondition:

Minimal Guarantees:

Success Guarantees:

Trigger:

Main Success Scenario:

Extensions:

Related Information:

❖ **תבנית חלקית ("casual")**

תבנית זו הינה פחות פורמלית ומתארת באופן מילולי בצורת סיפור את התרחיש הראשי וכן את ההרחבות כחלק מהסיפור. להלן התבנית :

Use Case #__ Use Case Name:

Primary Actor:

Scope:

Level:

< describe the use case as a short story with paragraphs. First paragraph for the main success scenario and the rest for the extensions >

08/01/09

❖ תבנית טבלאית

<u>תבנית עמודה אחת</u>:

Use Case #__	Use Case Name:	
Primary Actor		
Scope		
Level		
Stakeholders & Interests		
Precondition		
Minimal Guarantees		
Success Guarantees		
Trigger		
Description	Step	Action
	1	
	2	
	3	
Extensions	1a	
	1b	

<u>תבנית 2 עמודות</u>:

תבנית זו מתארת את ה Use Case בצורת "פינג-פונג" בין השחקן הראשי למערכת. בעמודה השמאלית יופיעו פעולות השחקן הראשי ובעמודה הימנית יופיעו תגובות המערכת.

Actor	**System**
Action #1	
	Response #1
	Response #2
Action #2	
	Response #1

❖ **תבנית ממוספרת**

תבנית זו נמצאת בשימוש במודל RUP[17]. התבנית משתמשת במספור מקונן ומכילה את מרבית המרכיבים של התבנית המלאה. להלן תבנית לדוגמא:

1. Use Case Name
   1.1. Brief Description
      … text …
   1.2. Actors
      … text …
   1.3. Triggers
      … text …
2. Flow of Events
   2.1. Basic Flow
      … text …
   2.2. Alternative Flows
      2.2.1. Condition 1
      … text …
      2.2.2. Condition 2
      … text …
      2.2.3. . . .
3. Special Requirements
   3.1. Platform
      … text …
   3.2. . . .
4. Preconditions
      … text …
5. Postconditions
      … text …
6. Extension Points
      … text …

[17] Rational Unified Process

2.5.9. שם ה Use Case:

שמו של ה Use Case יופיע בכותרת ה Use Case. שם הUse Case מהווה מזהה ייחודי עבור ה Use Case. השם צריך להכתב בצורת פעולה (לדוגמא: ביצוע הזמנה, משיכת מזומן) ועליו לתאר את המטרה להשגה. השם צריך להיות מספיק ברור עבור הקורא על מנת שיוכל להבין באופן כללי במה ה-Use Case עוסק. על השם להיות תמציתי ולא ארוך מדיי, בד"כ 2-3 מילים יספיקו. השם צריך להיות מונחה-מטרה (Goal-driven) בהתאם למטרת השחקן ובכך מכוון את כל אופן כתיבת הUse Case בצורה נכונה כאשר השחקן במרכז והפעולות נועדות להשגת מטרתו.

3.5.9. שחקן ראשי:

השחקן הראשי של ה Use Case הוא בעל העניין אשר מפעיל את המערכת על מנת לקבל שירות מסויים. ע"י פעולה זו השחקן הראשי מנסה לספק את מטרתו. בדרך כלל , השחקן הראשי הוא השחקן שמתניע (triggers) את ה-Use Case. ישנם שני מקרים אופייניים שבהם ה Use Case איננו מותנע באופן ישיר ע"י השחקן הראשי. מקרה ראשון הינו המקרה בו שחקן אחר (כגון פקיד או טלפנית) מתניעים את ה-Use Case בעבור השחקן הראשי. אולם, עם ההתקדמות הטכנולוגית, מקרה זה הולך ונהיה פחות שכיח וזאת מכיוון שלשחקן הראשי יש יותר ויותר אמצעים להתניע את ה-Use Case באופן עצמאי (למשל דרך ה-WEB , מערכת מענה טלפוני אוטומטי וכד'). בכל מקרה, חשוב להבין כי התנעת ה-Use Case במקרה זה מבוצעת באופן עקיף על ידי השחקן הראשי וה- Use Case בא על מנת לספק את מטרתו של השחקן הראשי ולא של אותו פקיד אשר מתניע אותו בעבורו. המקרה השני הינו המקרה בו הזמן הוא זה שמניע את ה-Use Case (למשל, ה-Use Case מותנע בכל יום בחצות). במקרה זה השחקן הראשי הינו בעל העניין אשר מעוניין בריצה של ה-Use Case בזמן זה.

השחקן ממלא תפקיד מסויים בעת האינטראקציה עם המערכת. הוא לא בהכרח אינדיבידואל מסויים. השחקן עשוי להיות מערכת אחרת או להיות מייצג של קטגוריה שלמה של אינדיבידואלים בעלי תפקיד זהה בהקשר של הפעלת המערכת (ב-Use Case המדובר).

השחקן הראשי חשוב בשתי נקודות זמן עיקריות במהלך התהליך: בתחילת התהליך (איסוף הדרישות) ובסיום התהליך (לקראת מסירת המערכת). בזמן שבין האירועים הללו , אין חשיבות גדולה למונח זה. בתחילת התהליך מבוצעת פעילות של זיהוי שחקנים. פעילות זו עוזרת לנו לאפיין את דרישות המערכת וזאת על ידי הבנת המטרות שכל שחקן רוצה להשיג מהמערכת. יצירת רשימה של שחקנים ראשיים בתחילת התהליך עוזרת לנו לאפיין את משתמשי המערכת, מהווה בסיס לרשימת שחקן-מטרה ותעזור לנו בהמשך כאשר נרצה לבצע חלוקה של Use Cases לחבילות (אשר סביר שיינתנו לפיתוח של

צוותים שונים). בסיום התהליך , השחקנים הראשיים שוב חשובים לנו מכיוון שייתכן ונרצה לבצע חלוקה לחבילות אשר ייטענו בהתאם למחשבי המשתמשים השונים. ייתכן ונרצה לקבוע רמות אבטחה שונות בהתאם לשחקנים השונים (לכל Use Case). כמו כן, סביר שנרצה לתכנן תוכנית אימונים לקבוצות משתמשים שונות.

4.5.9. Scope:

מונח זה למעשה מגדיר את גבולות המערכת בהקשר של ה-Use Case. חשוב מאוד להגדיר מה כלול בUse Case ומה מחוצה לו. טעות בהגדרה זו עשויה לעלות ביוקר – שינוי מאוחר ב Scope עשוי לגרור שינוי בתכולות הפרוייקט כולו וזה מתבטא בעלויות, לוח זמנים וכו'. נבחין בין שני סוגים של Scope :

- **Scope פונקציונלי:**

  Scope זה מתייחס לשירותים שהמערכת מספקת אשר בסופו של דבר ישוקפו בעזרת ה Use Cases. כאשר אנו בתחילת הפרוייקט אנו מחליטים על ה-Scope הפונקציונלי במקביל לתהליך זיהוי ה-Use Cases. ישנה השפעה הדדית בין שני התהליכים. על מנת להתמודד עם הבעיה ישנם מספר כלים אשר עוזרים לנו להתמקד בScope הנכון:

- רשימת In/Out : ברשימה זו ישנן 3 עמודות. עמודה ראשונה מציינת את הנושא הנדון , העמודה השנייה מציינת In והעמודה האחרונה מציינת Out. רשימה זו תבנה בדרך כלל במהלך פעילות של "סיעור מוחות" ותתעדכן במהלך הזמן. ברשימה זו קל לראות אילו נושאים אינם בתחום האחריות של המערכת (בד"כ המערכת תתממשק למערכת אחרת על מנת לבצע אותם).
- רשימת שחקן-מטרה : ברשימה זו ישנן 3 עמודות. עמודה ראשונה מציינת את השחקנים הראשיים, העמודה השנייה מציינת את מטרות השחקנים ביחס למערכת והעמודה האחרונה מציינת את העדיפות לספק מטרה זו ביחס ליתר המטרות במערכת. רשימה זו צריכה להתעדכן באופן דינמי והיא צריכה לשקף בכל זמן של פיתוח המערכת את הגבולות הפונקציונליים של המערכת.
- תקצירי Use Cases : בעוד רשימת שחקן-מטרה מהווה תיאור של התנהגות המערכת ברמת על, תקצירי Use Cases מהווים תיאור ברמת דיוק גבוהה יותר אך עם זאת עדיין מהווים תיאור כללי המתמקד אך ורק בתרחיש ההצלחה העיקרי. תקציר זה הינו תיאור של התנהגות ה-Use Case ב2 עד 6 משפטים המתארים את הפעילות המהותית. תקצירים אלו משמשים כאינדקס לUse Cases של המערכת ועוזרים להעריך את סיבוכיות תכולות העבודה.

- **Scope תכן :**

Scope תכן מגדיר למעשה את <u>היקף המערכת</u>. Scope תכן הוא ה-Scope אליו נתייחס כאשר אנו רושמים את סעיף ה-Scope בUse Case אותו אנו בונים. ה-Scope מגדיר את המערכות , החומרה והתוכנה אשר אנו אחראים לתיכון שלה. לדוגמא, אם אנו מפתחים כספומט , כל מה שימצא בקופסה של הכספומט כולל החומרה והתוכנה הינו ב-Scope שלנו, אולם רשת המחשבים איתה תדבר הקופסה איננה חלק מהתיכון שלנו ונמצאת מחוץ ל-Scope התכן. הגדרה מדוייקת של Scope זה מסייעת ליצירת שפה משותפת בין כל הקוראים/כותבים של ה- Use Case. Scope התכן מותאם לסוג ה-Use Case. *Business* Use Case זהו Use Case אשר ממדל תהליך עסקי ובמקרה זה ה-Scope הינו ברמת הארגון (Enterprise). עבור *System* Use Case אשר ממדל את המערכת ה-Scope יהיה מערכת המחשוב. עבור *Component* Use Case אשר ממדל מודול של המערכת ה-Scope יהיה תת-מערכת המכילה את אותו המודול.

Scope התכן מושפע ממסמך החזון[18] של המערכת. במסמך החזון ניתן לראות מה הניע את יוזמי הפרוייקט ,כיצד הם רואים את התפתחות המערכת בעתיד ומהן דרישות העל מהמערכת. כאשר נתקלים באי-בהירות ביחס לתכולה מסויימת , מקובל לחזור למסמך חזון המערכת ולנסות להבין מתוכו האם התכולה צריכה להכלל במערכת המפותחת או שמא היא מחוץ ל-Scope.

5.5.9. רמה:

Alistair Cockburn (Cockburn, 2000) מבחין בין שלוש רמות שונות של Use Cases :

- **רמת Summary Goals**

זו רמת העל במודל ובה אנו מספקים מטרות של מספר שחקנים. משך זמן ביצוע של Use Case ברמה זו הינו בדרך כלל בסדרי גודל של שעות , ימים , חודשים. בדרך כלל לא יהיו יותר מ 5-4 Use Cases ברמה זו (גם במערכות גדולות). Use Cases ברמה זו מסכמים את האינטרסים של מספר שחקנים ראשיים במערכת ומסייעים לנו בשלושה נושאים :

- מספקים קונטקסט ל-Use Cases ברמת ה-User Goals.
- מציגים את מחזור החיים של המטרות השונות.
- מהווים תוכן עניינים ל-Use Cases ברמות הנמוכות.

---

[18] Vision Document

Use Cases ברמה זו מכונים חיצוניים ביותר (outermost) והם אינם מסופקים לצוותי הפיתוח אשר עובדים על Use Cases ברמת User Goals.

- **רמת User Goals**

זו הרמה המרכזית במודל. ברמה זו השחקן הראשי או שחקן אחר מנסה לגרום למערכת לבצע עבודה מסויימת. משך זמן ביצוע של Use Case ברמה זו הינו בדרך כלל בסדרי גודל של דקות. ישנם מספר מבחנים אשר עוזרים בסיווג ה-Use Case לרמה המתאימה. ניתן לשאול את השאלות הבאות: "האם השחקן הראשי יכול ללכת מאושר לאחר סיום הביצוע?", "האם מעריכים את ביצועיו של השחקן לפי מספר הפעולות הללו שיבצע ביום?", "האם לאחר גמר ביצוע הפעולה, יכול השחקן לצאת להפסקת קפה?". השגה של המטרה ברמה זו משיגה מספר מטרות של רמה נמוכה יותר (Subfunctions). רשימה של Use Cases ברמה הזו מהווים למעשה תמצית של פונקציונליות המערכת כולה והם מהווים בסיס לחלוקות שונות כגון חלוקה לחבילות פיתוח, צוותים וכו'.

- **רמת Subfunctions Goals**

זו הרמה התחתונה במודל. אלו Use Cases שנדרשים בעיקר בשביל הקריאות[19] של המודל או בגלל ש-Use Cases אחרים משתמשים בהם. דוגמאות ל-Use Cases ברמה זו: חפש מוצר, חפש לקוח, שמור קובץ וכו'.

ככלל ניתן לומר כי ה-Use Cases העיקריים הינם מרמת Use-Goals ואת מרבית הזמן יש להשקיע באיתורם. מקצת ה-Use Cases שיש לכתוב יהיו מרמת Summary-Goals והם יספקו את הקונטקסט לאחרים.

6.5.9. בעלי עניין:

בעל ענין[20] זה מישהו או משהו שיש לו אינטרס מההתנהגות של ה-Use Case. הוא משתתף בחוזה. שחקן זה כל דבר שיש לו התנהגות. כל שחקן ראשי הוא בעל ענין, אולם לא תמיד כל בעלי הענין באים במגע ישיר עם המערכת אך למרות זאת יש להם זכות להשפיע כיצד המערכת תתנהג. לדוגמא: הבעלים של המערכת, חברי הדירקטוריון, גופי רגולציה וכדומה. ה-Use Case צריך להכתב בצורה שהאינטרסים של בעלי העניין נשמרים. שחקן ראשי של Use Case הוא אותו בעל ענין אשר מפעיל את המערכת על מנת לקבל את אחד משירותיה על מנת להשיג את מטרתו. בדרך כלל השחקן הראשי הוא זה שיוזם את הפעלת ה-Use Case.

---

[19] Readability

[20] Stakeholder

7.5.9. תנאים מקדימים[21]:

אלו התנאים שמובטח על ידי המערכת כי יתקיימו בטרם הפעלת ה-Use Case. מכיוון שתנאים אלו מובטחים הם לא נבדקים שוב במהלך ריצת ה-Use Case. דוגמא נפוצה היא שהמשתמש נכנס למערכת ונבדק כי הוא אכן רשאי להכנס.

8.5.9. תנאים מאוחרים[22] מינימליים:

אלו התנאים המועטים אשר מובטחים לבעלי הענין כי יתקיימו גם כאשר המטרה של השחקן הראשי לא מתקיימת. תנאים אלו מתקיימים בכל מקרה בסיום של כל הרצת Use Case גם אם המטרה הושגה וגם אם לאו. תנאי מאוחר מינימלי לדוגמא : המערכת תרשום ביומן האירועים עד לאן הגיעה. גם במקרה של כשלון מובטח לנו כי ישנו קובץ log שניתן לקרוא מתוכו אילו שלבים בוצעו עד לכשלון.

9.5.9. תנאים מאוחרים בהצלחה:

אלו התנאים אשר יתקימו רק במקרה של הרצה מוצלחת של ה-Use Case (בתרחיש ההצלחה הראשי או בתרחיש הצלחה אלטרנטיבי). הרצה מוצלחת היא הרצה שבסיומה המטרה של בעל הענין מושגת ובמקרה זה התנאים המאוחרים בהצלחה מתווספים לתנאים המינימליים אשר הובטחו לנו מראש.

01.5.9. טריגר:

הטריגר מייצג את אותו אירוע אשר גורם להתנעת ה Use Case. לעיתים הטריגר מופיע כשלב מקדים לצעד הראשון ולעיתים הוא נכתב כצעד ראשון של ה Use Case. במקרים רבים הטריגר הינה פעולה שמבצע השחקן הראשי אך יתכן מקרה כי הטריגר הינו אירוע שמותנה בזמן (לדוגמא : גיבוי אוטומטי יומי).

11.5.9. תרחיש הצלחה ראשי:

את תרחיש ההצלחה הראשי יש לכתוב ראשון והוא מתאר מלמעלה למטה בצורה קריאה וברורה את התרחיש הטיפוסי שבו מטרת השחקן סופקה והאינטרסים של בעלי הענין הושגו. תרחיש ההצלחה הראשי מתואר כסדרת צעדים הכוללים פעולות שנועדו להשגת המטרות של השחקנים השונים. הצעדים נכתבים בצורה סדרתית אולם יתכן ויתקיימו מספר פעולות במקביל , תתכן חזרה ויתכן כי צעדים מסויימים הינם אופציונליים בלבד. באופן כללי ניתן לומר שצעד בתרחיש מתאר :

---

[21] Pre Conditions

[22] Post Conditions

- אינטראקציה בין 2 שחקנים
- צעד אימות אשר נועד לשמור על אינטרס של בעל ענין
- שינוי פנימי שנועד לצורך סיפוק אינטרס של בעל ענין

21.5.9. הרחבות[23]:

הרחבות אלו הסתעפויות מתוך תרחיש ההצלחה הראשי אשר יתארו כשלונות או תרחישי הצלחה אלטרנטיביים. בכל נקודה שבה ההתנהגות עשויה להתפצל בעקבות תנאי מסויים יש לרשום את התנאי ואת הצעדים לטיפול בו. הרחבות רבות מתמזגות בהמשך חזרה עם תרחיש ההצלחה הראשי. להלן דוגמא למבנה של הרחבה :

. . .

4. User has the system save the work so far.

. . .

Extensions:

4a. System autodetects the need for an intermediate save:

4a1. . . .

4b. Save fails:

4b1. . . .

הרחבה זו מתארת למעשה מקרה בו המערכת עשויה לזהות את הצורך לבצע שמירה של מה שבוצע עד כה (autosave) ובמקרה זה היא תבצע משהו (את 4a1 וכו'). השמירה (אם ע"י בקשה מפורשת של המשתמש ואם ע"י autosave) עשויה להכשל ובמקרה זה המערכת תגיב בצורה מסוימת (ע"פ 4b1 וכו').

[23] לעיתים נקרא גם וריאציות / הסתעפויות / אלטרנטיבות / חריגות.

6.9. תרשימי Use Case

שפת UML תומכת בתרשים מסוג Use Case. תרשים זה מסייע בהבנת גבולות המערכת, בהכרת היישויות השונות והיחסים ביניהן. טעות נפוצה הינה שימוש בתרשים Use Case מבלי לצרף את תבנית ה-Use Case הכוללת תיאור מילולי מפורט של ה-Use Case. ללא תיאור מילולי מפורט לא ניתן לדעת מה קורה ב-Use Case והתרשים למעשה נותר חסר-ערך. תרשים ה-Use Case נותן לנו סקירת על תמציתית של התנהגות המערכת.

להלן פירוט האלמנטים השונים המרכיבים את תרשים ה-Use Case :

- שחקן :

איור30

מיוצג ע"י ציור של "איש-מקל" ומתחתיו רישום של שם השחקן. יש הטוענים כי ייצוג זה איננו טוב ועשוי להטעות וזאת מכיוון ששחקן איננו בהכרח אדם ועשוי לייצג מערכת חיצונית.

- Use Case :

איור31

מיוצג ע"י אליפסה ובתוכה רשום שמו של ה Use Case.

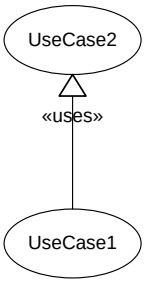
- יחסים :
  - יחס בין שחקן ו Use Case :

איור32

מיוצג ע"י קו מחבר בין השחקן לבין בועת ה Use Case. כל שחקן המופיע ב Use Case יחובר באמצעות קו לבועת ה Use Case.

- יחס בין שני Use Cases :

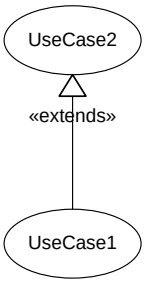
יחס הכלה (<<Uses>> או <<Include>>)



איור33

מיוצג ע"י קו עם ראש חץ בין ה-Use Cases וציון של Stereotype של <<uses>> או <<include>> ע"ג הקו. יחס זה מציין כי Use Case אחד מבצע שימוש ב Use Case שני (אחד מצעדי ה Use Case הראשון מריץ את הUse Case השני).

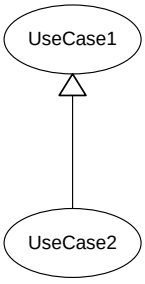
יחס הרחבה (<<Extends>>)



איור34

מיוצג ע"י קו עם ראש חץ בין ה-Use Cases וציון של Stereotype של <<extends>> ע"ג הקו. יחס זה מציין כי Use Case אחד הינו הרחבה של ה-Use Case השני. ניתן להשתמש ביחס זה גם כאשר הפעלת ה-Use Case הנוסף הינה אופציונלית.

יחס הכללה (<<Generalization>>)



איור35

מיוצג ע״י קו עם ראש חץ בין ה-Use Cases ללא ציון של Stereotype . יחס זה מציין כי Use Case אחד הינו הכללה של ה-Use Case השני , בדומה ליחס ירושה בין מחלקות.

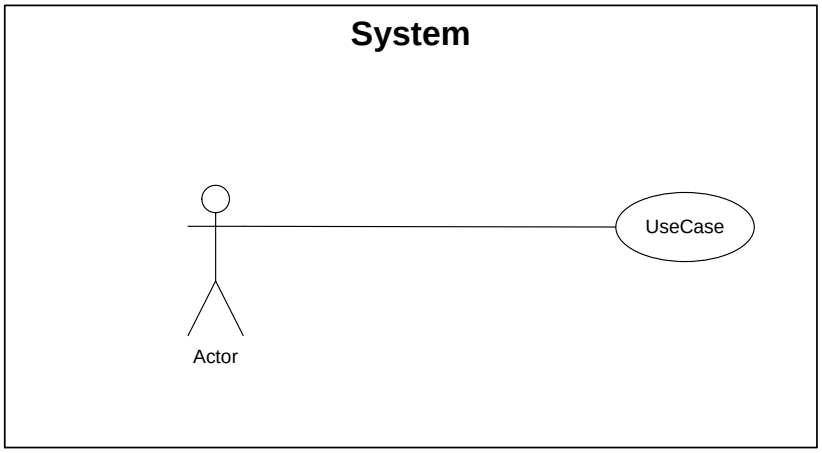
- יחס בין שני שחקנים :

יחס הכללה (<<Generalization>>)

איור36

מיוצג ע״י קו עם ראש חץ בין השחקנים ללא ציון של Stereotype . יחס זה מציין כי שחקן אחד הינו הכללה של השחקן השני , בדומה ליחס ירושה בין מחלקות.

- גבולות המערכת :

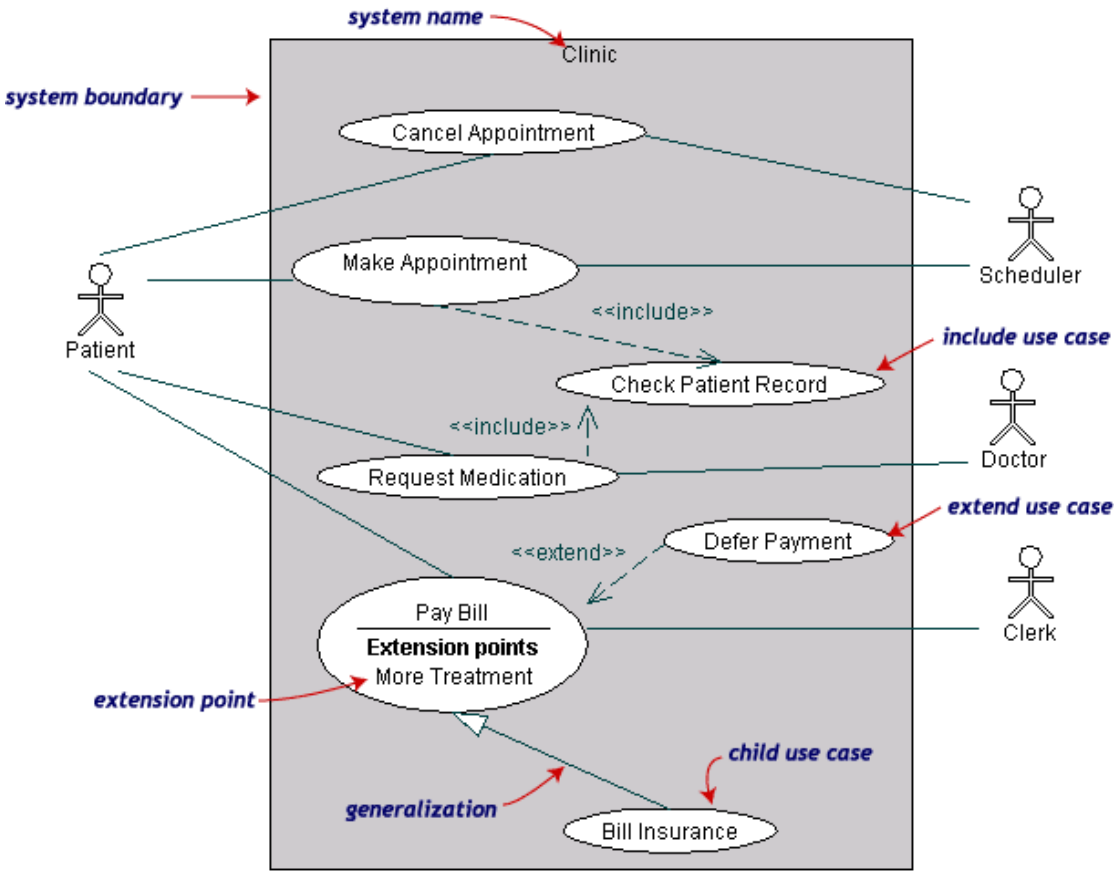


איור37

מיוצג באמצעות מלבן המכיל את תרשים ה-Use Case. ניתן לציין את שם המערכת בראש המלבן.

08/01/09

- תרשים Use Case לדוגמא:



איור38

רשימת מקורות :

[1] Anda, B. and Sjoberg, D., "Towards an Inspection Technique for Use Case Models", 14th Int.Conf. on Software Engineering and Knowledge Engineering (SEKE'02), Ischia, Italy, 2002. <http://portal.acm.org/citation.cfm?id=568785&dl=acm&coll=&CFID=15151515&CFTOKEN=6184618>

[2] Armour, F. and Miller, G. "Advanced Use Case Modelling.", Addison-Wesley, 2000.

[3] Berard, E., "Be Carful with Use Cases", <http://www.toa.com/pub/use_cases.htm>, 1998.

[4] Biddle, R., Noble, J. and Tempero, E. "Essential Use Cases and Responsibility in Object-Oriented Development", <http://www.foruse.com/articles/euc-responsibility.htm>, 2001.

[5] Bittner, K. and Spence, I. "Use Case Modeling.", Addison-Wesley Professional, 2002.

[6] Cockburn, A. "Writing Effective Use Cases.", Addison-Wesley, 2000.

[7] Cockburn, A., "Structuring Use Cases with Goals: Parts 1 and 2," Journal of Object Oriented Programming, Sept - Oct 1997 and Nov - Dec 1997. < http://atlas-project-tdaq-awg.web.cern.ch/atlas-project-tdaq-awg/Documents/StructuringUseCasesWithGoals.pdf>

[8] Cockburn, A., Highsmith, J., "Agile Software Development", Addison-Wesley, 2006.

[9] Gutierrez, J. , Escalona, M. and Torres, L., "An Approach to Generate Test Cases from Use Cases" , ACM International Conference Proceeding Series; Vol. 263, 2006.

[10] Heeseok, C., Keunhyuk, Y., "An approach to software architecture evaluation with the 4+1 view model of architecture", Software Engineering Conference, Ninth Asia-Pacific, 2002.

[11] Jagielska, D., Wernic, P., Wood, M. and Bennett S., "How Natural is Natural Language? How Well do Computer Science Students Write Use Cases?" , Dynamic Languages Symposium:Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications , 2006.

[12] Kruchten P., "The Rational Unified Process: An Introduction, Third Edition", Addison-Wesley, 2003.

[13] Kulak D., Guiney E., "Use Cases: Requirements in Context", Addison-Wesley, 2000.

[14] Leffingwell D., Widrig D., "Managing Software Requirements, Second Edition", Addison Wesley, 2005.

[15] Maiden, N. and Robertson, S. "Developing Use Cases and Scenarios in the Requirements Process", Software Engineering ICSE Proceedings. 27th International Conference", 2005.

[16] Malan, R. and Bredemeyer, D., "Functional Requirements and Use Cases", Bredemeyer Consulting, 2001.
<http://www.bredemeyer.com/pdf_files/functreq.pdf>

[17] Mills, D., "What's the Use of a Use Case?"
<http://www.softed.com/Resources/WhitePapers/UseCase.aspx>, 2002.

[18] Mohagheghi, P., Anda, B. and Conradi, R., "Effort Estimation of use cases for incremental large-scale software development", Software Engineering ICSE 2005. Proceedings. 27th International Conference , 2005.

[19] Object Management Group, OMG Unified Modelling Language Specification V1.3, June 1999: < http://www.jeckle.de/uml_spec.htm >

[20] Övergaard G., Palmkvist K., "Use Cases Patterns and Blueprints", Addison Wesley Professional, 2004.

[21] Rosenberg, D., Scott, K., "Use Case Driven Object Modeling with UML: A Practical Approach", Assidon-Wesley, 2001.

[22] Rosenberg, D., Stephens, M., "Agile Development with ICONIX Process", Apress, 2005.

[23] Schneider G., Winters P.J., "Applying Use Cases, Second Edition.", Addison Wesley, 2005.

[24] Savransky D., "Engineering of Creativity: Introduction to TRIZ Methodology of Inventive Problem Solving", CRC Press, 2000.

[25] Wiegers K.E., "More About Software Requirements", Microsoft Press, 2006.