פרק 1.4 ■ גישות לפיתוח מערכות מידע

גישת פיתוח היא השיטה הכללית (או האסטרטגיה) שלפיה מתנהל פרויקט הפיתוח, כלומר, הסדר הכללי שמבצעים בו את שלבי פיתוח המערכת. גישת פיתוח אינה מכתיבה נוהלי עבודה ספציפיים וגם לא מתודולוגיית פיתוח או שיטות וטכניקות שאמורים להשתמש בהם בשלבי הפיתוח.[11]

בפרק זה יוצגו גישות הפיתוח העיקריות שהתפתחו במשך השנים: גישת "בנה ותקן", מודל "מפל המים", גישת האב־טיפוס, המודל הספירלי ופיתוח תוספתי־מחזורי. גם פרק 5 ימשיך בנושא זה ויעסוק בגישת פיתוח "זריזה" (Agile) ובאחת השיטות הנפוצות השייכת לגישה זו – שיטת Scrum.

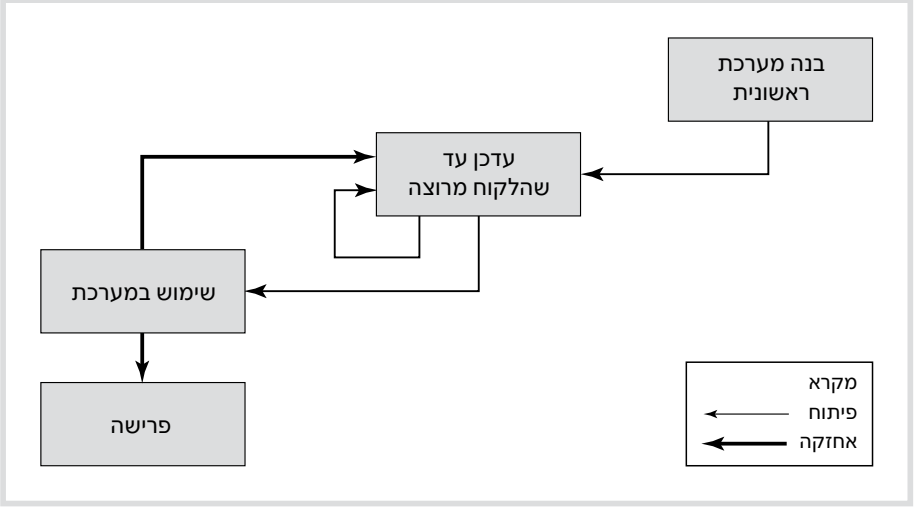
לפני פירוט הגישות השונות נציין שקיים עוד בלבול בין שני מונחים: בחלק מהמקרים משתמשים במונח "גישת פיתוח" ובחלק במונח "מודל פיתוח". למען הדיוק המונח הנכון הוא "גישת פיתוח" שהרי מודל אינו גישה אלא אמצעי לייצוג המציאות או המערכת. לדוגמה: מודל פונקציונלי של מערכת מבוטא באמצעות תרשים DFD; מודל נתונים של מערכת מבוטא באמצעות תרשים ERD. למרות זאת יש מקרים שבהם במקום "גישה" משתמשים במונח "מודל" (מכיוון שכך כינה אותו ממציא הגישה/המודל או מכיוון שכך השתרש המונח). לפיכך נשתמש גם אנו במונח המקובל לכל גישה.

1.4.1 גישת "בנה ותקן"

גישת "בנה ותקן" לפיתוח מערכות תוכנה היא גישה "עתיקה" (משנות ה־60 של המאה ה־20) שעל־פיה פיתחו מערכות מידע (מבלי לכנותה בשם זה) לפני שהוכר בחשיבותה של הנדסת התוכנה, לפני שנעשתה הבחנה בין שלבים שונים בתהליך הפיתוח ולפני שפותחו מרב השיטות והטכניקות לפיתוח מערכות ותוכנה. כשהוחל לפתח מערכות

11 לא תמיד מבחינים בין המונחים מתודולוגיית פיתוח, נוהל פיתוח וגישת פיתוח; יש המתייחסים לגישת פיתוח כאל מתודולוגיה ויש הכורכים את שניהם ואף את המונח נוהל פיתוח מבלי להבחין ביניהם. אף שמדובר בעיקר בסמנטיקה אנו ניצמד להבחנה שנעשתה כאן בין המונחים.

בגישה זו, הטכניקות העיקריות שהיו ידועות היו כתיבת תכניות מחשב בשפות התְכנות שהיו קיימות אז (כגון COBOL ו־FORTRAN), ובפרט העקרונות של תכנות מובנה (structured programming). ואולם לא היו אז שיטות וטכניקות לביצוע שלבים קודמים של תהליך הפיתוח ובפרט ניתוח המערכת ועיצובה. בגישת "בנה ותקן" המערכת מפותחת ומוטמעת בארגון בלי שהוגדרו בבירור צורכי הלקוח. המפתחים, שהם בעיקרו של דבר מתכנתים, בונים (כלומר מתכנתים) מערכת ראשונית על־פי דרישות ראשוניות שהלקוח מציג, ולאחר הטמעתה אצל הלקוח הם מתקנים ומשפרים אותה עד שהיא עונה על דרישותיו. תיאור סכמתי של המודל מוצג בתרשים 1.8.



תרשים 1.8: גישת "בנה ותקן".

לכאורה זוהי גישת פיתוח פסולה שהתאימה לתקופה הראשונית של עולם התוכנה, לפני שהומצאו גישות אחרות. הגישה פסולה בין היתר מכיוון שאין מגדירים היטב את הדרישות, אין משתפים מספיק את המשתמשים בתהליך הפיתוח, ממהרים לתכנת ולהתקין את המערכת מבלי לבחון אותה, אין תיעוד של המערכת שפותחה, אין תכנון ובקרה על הפיתוח, אין הקפדה על עמידה בלוחות זמנים ובתקציבים וצפוי שיהיה צורך לחזור ולתקן את המערכת עד שתשביע את רצון המשתמשים, אם בכלל.

למרות כל זאת, בנסיבות מסוימות אפשר למצוא בגישה זו גם יתרונות: מנקודת מבטם של המשתמשים הם מקבלים מערכת מידע מתפקדת (גם אם לא טובה) בשלב מוקדם. מנקודת מבטם של המתכנתים אפשר להניח שהם מרוצים מגישה זו מכיוון שבאופן טבעי

הם מעדיפים לתכנת ולא לעסוק בתכנון המערכת ובניתוחה וגם לא להיות נתונים לבקרה. גישה זו עשויה להיות יעילה גם בימינו כשמדובר ב**מערכת קטנה יחסית ושאינה קריטית לארגון**, אך ברור שלא רצוי לפתח על־פיה מערכת גדולה ומורכבת שכן עלולה להתקבל מערכת שלא תענה על הצרכים האמתיים של הלקוח ושעלות התיקון של שגיאותיה תהיה גבוהה. גם אחזקת המערכת תהיה קשה בשל היעדר מסמכי תכנון.

על אף ההסכמה על מגבלותיה יש גורמים אחדים שבגללם גישת "בנה ותקן" מיושמת לעתים בארגונים. נמנה אחדים מהם:

גורם פסיכולוגי: ממחקרים מתברר שאחת התכונות המאפיינות אנשי פיתוח תוכנה היא צורך חזק באתגרים ובהישגיות. פיתוח תוכנה מספק צורך זה מכיוון שבתהליך התכנות אפשר לראות התקדמות ותוצאות החל בכתיבת תכניות המכילות שגיאות וכלה בתכניות מתפקדות ונטולות שגיאות. הקדשת זמן בסדר גודל דומה ללימוד דרישות המשתמשים ולאפיון מערכת המידע אינה מספקת תוצאות והישגים כה מובהקים ומכאן הנטייה להימנע מכך. יש הטוענים שלאנשי פיתוח תוכנה יש צורך חברתי נמוך (low social need); הם מעדיפים לעבוד לבד ולא בצוות עם משתמשים שאין להם שפה משותפת אתם כפי שנדרש בעבודת האפיון והניתוח של מערכת מידע.

גורם החינוך: במהלך הכשרתם במוסדות ההשכלה השונים אנשי פיתוח תוכנה עוסקים בפתרון בעיות דטרמיניסטיות, דהיינו בעיות מוגדרות היטב שאינן מצריכות התלבטויות רבות ויש להן פתרון ברור. בעצם פתרון של תרגיל כלשהו בתכנות מדגים את גישת "בנה ותקן": הלומד מקבל דרישות בצורת מלל כלשהו, ופתרון הבעיה מתבטא בעיקר בתכנות ובתיקון עד שמתקבלת תכנית שמתפקדת. אדם שיוצא מסביבה לימודית ומתקבל לעבודה מעשית בתחום פיתוח התוכנה שָׂשׂ לתכנת ולא לעסוק בלא נודע, כלומר באפיון צרכים ובניתוח מערכת לפני כתיבת תכניות המחשב.

גורם אופי התוכנה: הרבה מערכות נראות פשוטות לכאורה לפני שניגשים לפתח אותן. אחת הסיבות לכך היא שמערכת איננה ישות מוחשית (tangible), אי־אפשר "לראות" אותה. לכאורה נראה שקל לעמוד במשימת הפיתוח בלי תכנון וניתוח מכיוון שקשה יותר לזהות מה בדיוק כרוך בעמידה בדרישות. שפות תכנות מתקדמות נוטות להחמיר את הבעיה מפני שהן מעודדות עוד יותר יישום (תכנות) נמהר של המערכת. רק לאחר שמתחילים בתהליך הפיתוח מסתבר שהמערכת אינה פשוטה וצצות בעיות שונות שלא נראו ולא נבחנו מלכתחילה.

לסיכום, פיתוח מערכת מידע בגישת "בנה ותקן" לא הוכיח את עצמו. יישום הגישה במערכות מידע גדולות פוגע בדרך כלל באמינות אנשי הפיתוח מבחינת יכולתם לנהל את משימת הפיתוח, להעריך את עלויותיה, לעמוד בלוח הזמנים ולהנפיק תוצר איכותי.

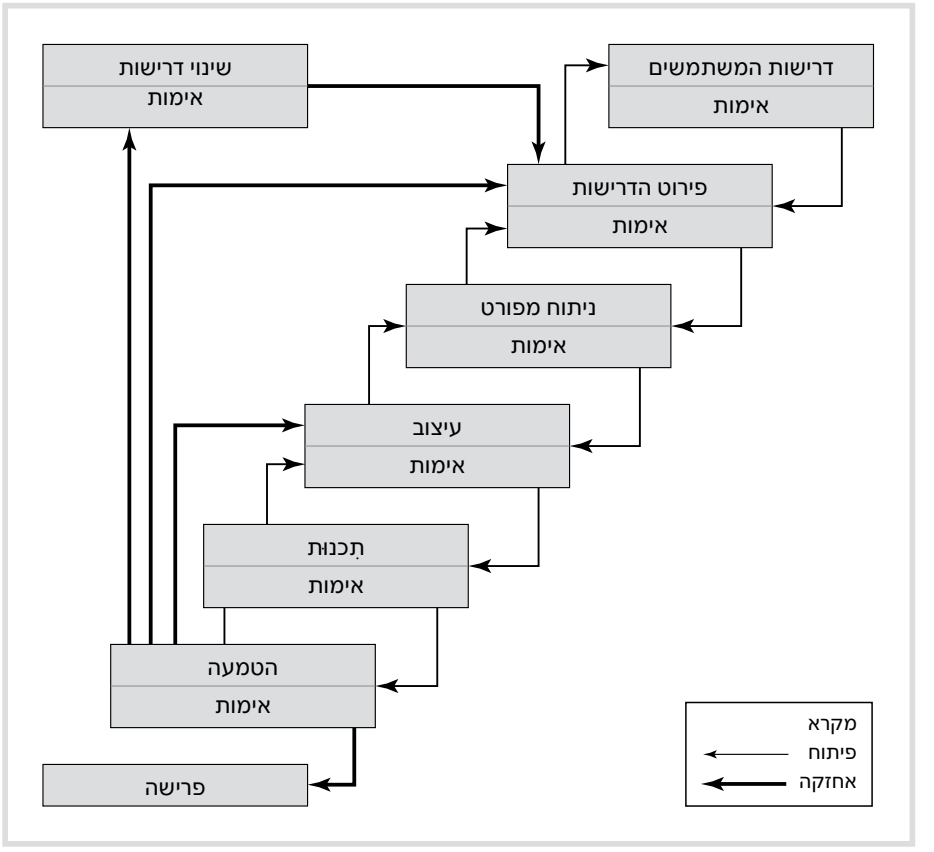
1.4.2 מודל "מפל המים"

מודל "מפל המים" (waterfall) שפיתח Royce בשנת 1970 היה המודל השליט בפיתוח מערכות מידע במשך שנים רבות ומקובל גם כיום. גישת פיתוח זו פותחה כמענה לבעיות שהתעוררו בגישת "בנה ותקן". זוהי הגישה הראשונה שהבחינה בין שלבי הפיתוח והדגישה את הצורך לבצע כל שלב במלואו ובשיטות עבודה מובְנות בטרם ממשיכים לשלב הבא. משמע, תהליך הפיתוח הוא בעיקרון **סדרתי** (לינארי) מתחילת הפרויקט ועד סופו. עם זאת, כל שלב גם יכול לעדכן את השלב שקדם לו במידת הצורך. הגישה מוצגת בתרשים 1.9.

לפי גישה זו, בצוות הפיתוח משתתפים הן אנשי מקצוע (מפתחי מערכת המידע) והן נציגי המשתמשים במערכת ובראשם צוות ניהול פרויקט הפיתוח. פיתוח המערכת מתחיל בהגדרת צורכי המשתמשים על־פי מסמך דרישות כתוב. לאחר אימות המסמך ואישורו ממשיכים לשלב הבא – פירוט הדרישות – שבו מתכננים את פרויקט הפיתוח על תקציבו ולוח הזמנים המפורט שלו. לאחר בדיקת התכנית ואישורה עוברים לשלב הבא – ניתוח והגדרה מפורטת ומדויקת ככל האפשר של המערכת שתפותח. מבלי לפרט את יתר שלבי הגישה (ראו פרטים בתרשים 1.9) נדגיש שכל שלב מסתיים בהפקת מסמך המתעד את מה שנעשה בו, וצוות הניהול צריך לאמתו ולאשרו; רק לאחר מכן עוברים לשלב הפיתוח הבא. החצים החוזרים מכל שלב לשלבים הקודמים מציינים שבעת הביצוע של כל שלב אפשר לשוב ולעדכן את מסמכי השלבים הקודמים לפי הצורך, שהרי ייתכן שבזמן שמבצעים שלב כלשהו מתגלים דברים שלא נודעו קודם או שהדרישות משתנות. (החץ החוזר משלב ההטמעה מציין שגם בשלב הטמעת המערכת ולאחריו ייתכן שיהיה צורך לחזור ולתקן את תוצרי השלבים הקודמים.) בגישה זו מסמכי פיתוח המערכת מעודכנים ותקפים כל העת.

מודל מפל המים נחשב בעבר להצלחה רבה בפיתוח מערכות מידע, וארגונים רבים אימצו אותו במשך שנים. לאור גישה זו פותחו נהלים ותקנים מפורטים הן מטעם ארגוני תקינה לאומיים ובין־לאומיים והן מטעם ארגונים שעסקו בפיתוח מערכות, ונהלים ותקנים אלה הגדירו במפורט את שיטת ניהול הפרויקט בגישה זו במטרה להבטיח יישום נכון ולאפשר בקרה עליו.

למרות יתרונותיה הברורים של גישת "מפל המים" בהשוואה לגישת "בנה ותקן" שקדמה לה, הסתבר שיש לה גם חסרונות. החיסרון העיקרי הוא הארכת זמן הפיתוח; הגישה מחייבת לסיים כל שלב לפרטיו ובמלואו לפני המעבר לשלב הבא, לקיים פגישות עבודה פורמליות כדי לדון בתוצרי כל שלב ולאשר את המעבר לשלב הבא, לתעד ולעדכן תיעוד



תרשים 1.9: מודל "מפל המים".

קיים בעקבות כל שינוי. כל אלה נועדו כמובן להבטיח פיתוח איכותי אך הדבר כרוך כאמור בהארכת זמן הפיתוח ולפיכך בהוספת כוח אדם ובעלויות נוספות אחרות. אי־אפשר לטעון שתוספת הזמן והמחיר אינם מצדיקים את המטרה של פיתוח איכותי כשמדובר במערכת מידע גדלה וחיונית, אבל במערכת שאינה כזאת אפשר אולי לוותר על ביצוע כל שלבי המשנה או על הפקת כל תוצרי הביניים כגון תיעוד מלא של תוצרי כל השלבים, ולחסוך בזמן הפיתוח ובעלויות.

בעיה חשובה במיוחד הנובעת מהארכת זמן הביצוע של פרויקט גדול היא שבמשך הזמן הארוך עשויים לחול שינויים רבים בדרישות המשתמשים כמו גם בטכנולוגיה ובסביבה העסקית, מה שמחייב ביצוע שינויים נוספים במהלך הפיתוח. ככל שחולף הזמן מצטברות דרישות לשינויים נוספים והיענות להם גוררת שוב דחייה בהשלמת הפיתוח וחריגות

בתקציב שתוכנן. נוסף על כך, בתקופת הפיתוח הממושכת נוצר נתק בין המשתמשים למפתחים; הם אינם מקבלים "מוצר מתפקד" כפי שהובטח להם ועלולים לאבד את אמונם בצוות הפיתוח ובסיכוי לקבל תוצר מוגמר.

לסיכום, על אף יתרונותיה של גישת "מפל המים", מחקרים וסקרים שבוצעו על פרויקטים רבים שפותחו בגישה זו במשך השנים הראו שהם לקו בחריגות גדולות בתקציבים ובלוחות זמנים. מצב זה גרר צורך למצוא פתרונות נוספים.

1.4.3 גישת ה"אב־טיפוס"

1.4.3.1 מהו אב־טיפוס בתוכנה?

המונח יצירת "אב־טיפוס" (prototype) שאול מתחומי ההנדסה והטכנולוגיה, שם יוצרים אב־טיפוס כחלק סטנדרטי בתהליך הפיתוח של מוצרים. יש יתרון גדול לבניית אב־טיפוס לפני ייצור המוצר עצמו: חברה שמעוניינת לפתח ולייצר מוצר בכמויות גדולות (לדוגמה מטוסים או מכוניות) צריכה להקצות משאבים יקרים הן לפיתוחו והן לייצורו ההמוני. כדי להבטיח מוצר איכותי שעונה על דרישות הלקוחות ומיוצר ביעילות בונים אב־טיפוס – מוצר ראשוני המשמש לבחינת המוצר בהשוואה לדרישות ולתכנונים. בעקבות בדיקתו מעדכנים (משפרים) את תכונות המוצר ואת תכניות הייצור ושוב מייצרים אב־טיפוס. התהליך יכול לחזור כמה פעמים עד שהאב־טיפוס הסופי עובר את מבחני הקבלה שנקבעו לו, ואז עוברים לייצור המוצר בהיקף מלא.

גישת האב־טיפוס אומצה לקראת שנות ה־80 כשאנשי תוכנה חיפשו פתרונות לבעיות שהתגלו בגישת מפל המים. הם הניחו שאם גישת האב־טיפוס מועילה לפיתוח מוצרים הנדסיים אחרים, סביר שתועיל גם לפיתוח תוכנה. ואכן המונח "אב־טיפוס בתוכנה" נעשה שגור מאז בפי רבים אך יש לו פירושים שונים. לפי אחת ההגדרות אב־טיפוס הוא מודל ראשוני של מערכת או של חלקים ממנה שנועד להדגים אותה באופן פונקציונלי. מטרתו העיקרית היא להעלות את הוודאות שפיתוח המערכת יצליח (למשל על־ידי קבלת אישור מוקדם מהמשתמשים שהמודל תואם את דרישותיהם). הדגם שאושר ישמש בסיס להמשך פיתוח המערכת. הגדרה זו אינה מדויקת דיה; לדוגמה: האם מודל ראשוני כזה אמור רק להדגים תכונות מסוימות של המערכת או שעליו להיות תוכנה מתפקדת ממש? הסיבה העיקרית לאי־הבהירות של המונח אב־טיפוס בתוכנה היא שפיתוח וייצור של תוכנה שונה מפיתוח וייצור בתחומי הנדסה "רגילים". שלא כמו מוצרי חומרה רבים, תוכנה אינה מיוצרת בתהליך תעשייתי אלא מפתחים אותה פעם אחת ולבסוף מתקבל המוצר (המערכת) שמותקן אצל הלקוח. במוצרים שמצריכים ייצור תעשייתי חלק ניכר מעלות המוצר טמון בהצטיידות במכונות ייצור, כלי עבודה וחומרי גלם ובהפעלת המפעל. האב־טיפוס משמש בתעשיות הללו לבדיקת מידת השימושיות, המעשיות או יכולת הייצור של המוצר לפני שעוברים לייצורו במלוא הכושר. בתוכנה לעומת זאת אין כאמור תהליך ייצור ואין (כמעט) חומרי גלם. המשימה היקרה, המסוכנת והממושכת ביותר היא הפיתוח החד־פעמי. הדילמה היא אם להשקיע בפיתוח אב־טיפוס של התוכנה במקום בפיתוח ובייצור חד־פעמי של המוצר עצמו.

יש שראו באב־טיפוס גישה חדשה לפיתוח תוכנה שמחליפה את גישת מפל המים ואחרים ראו בה תוספת או שיפור לגישה זו. למעשה נוצרו כמה סוגים של אב־טיפוס בתוכנה הנבדלים ביניהם במידת קרבתם למערכת המידע האמתית: מצד אחד יש אב־טיפוס שהוא דגם בלבד (mock up), שאינו אלא תצוגה של מסכי קלט ופלט שאין "מאחוריהם" דבר (כלומר אין תוכנה מתפקדת), ומהצד האחר יש אב־טיפוס שהוא מערכת מידע אמתית שמותקנת לתקופת ניסיון אצל הלקוח. בין שני אב־טיפוסים קיצוניים אלה יש עוד כמה סוגים של אב־טיפוס וכל סוג משפיע אחרת על תהליך הפיתוח הכולל של המערכת. בטרם נכיר את הסוגים השונים (בסעיף הבא) נבהיר את היתרונות הכללים הצפויים משימוש באב־טיפוס בתוכנה, ללא תלות בסוגו.

1.4.3.2 יתרונות גישת האב־טיפוס

- אב־טיפוס מפותח במהירות, בשלב מוקדם של תהליך הפיתוח, בהשקעה קטנה יחסית, והוא מאפשר להראות למשתמשים בשלב מוקדם יחסית מודל של המערכת שכולל לפחות את צורתה החיצונית – המנשקים, הקלטים והפלטים. (מכאן ואילך

נכנה את כל המרכיבים האלה "מנשקים".) לכן רבים משתמשים במונח "אב־טיפוס מהיר" (rapid prototype).

- אב־טיפוס הוא אמצעי יעיל לשיפור התקשורת בין המשתמשים למפתחים. לעתים המשתמשים מתקשים להבין את צורכי המידע שלהם או להבהיר את צורכיהם למפתחים, והמפתחים לא תמיד מבינים את הבעיות והצרכים שהמשתמשים מעלים בפניהם. פיתוח אב־טיפוס יוצר "שפה משותפת"; זהו מעין מוצר מוחשי שאפשר לראותו ולהבינו יותר מאשר אמצעים אחרים כגון מסמכים ותרשימים.
- אב־טיפוס מקנה חיזוק ותמריץ למשתמשים. הודות לשיפור התקשורת בין המשתמשים למפתחים המשתמשים ממלאים תפקיד פעיל יותר בפיתוח המערכת; הם יכולים להיווכח שרעיונותיהם מתורגמים במהירות לתהליך הפיתוח. הדבר גורם להם לתחושת שותפות ובעלות על המערכת. במקום המתנה ארוכה ומתסכלת עד לגמר הפיתוח המשתמשים רואים את המוצר שהם שותפים ליצירתו "גדל מול עיניהם". מובן שכך גדל הסיכוי לקליטה מוצלחת של המערכת.
- אב־טיפוס מקנה להנהלה יכולת שיפוט. בדומה ליתרונות הניתנים למשתמשים השותפים לתהליך הפיתוח, גם המנהלים יכולים "לראות" את המוצר המתפתח ולהעריכו לפני קבלת החלטות חשובות על המשך הפיתוח. הם גם יודעים שהמשתמשים המשתתפים בתהליך הפיתוח מבינים יותר מה הם עשויים לקבל ומה עוד עומד לפניהם. כל אלה מקטינים את הסיכון ומקלים את קבלת ההחלטות באשר להמשך השקעת המשאבים בפיתוח המערכת.
- אב־טיפוס מאפשר להנמיך ציפיות לממדים מציאותיים. כל עוד הדרישות והתכנונים הם "על הנייר" אי־אפשר לחוש את היקף משימת הפיתוח. יצירת אב־טיפוס מאפשרת לכל הצדדים – מפתחים, משתמשים ומנהלים – להתרשם מגודל הבעיה וממורכבותה. האב־טיפוס יכול לסייע במיתון ציפיות גבוהות מדי ולהצביע על סכנות בהמשך הדרך. הגישה מקנה אפשרות טבעית של ניסוי וטעייה בפיתוח מערכות מורכבות.
- אב־טיפוס מאפשר למשתמשים ולמפתחים לבחון רעיונות וחלופות באופן מעשי. אילולא גישה זו בחינת חלופות שונות הייתה תאורטית בלבד ואילו בפועל היו מפתחים רק את הגרסה שנבחרה מראש. אחת האפשרויות בגישת האב־טיפוס היא קודם לפתח ולבחון דרכים שונות (כלומר אב־טיפוסים שונים) במהירות ובהשקעה קטנה יחסית, ורק אחר כך להשקיע בפיתוח הפתרון הנבחר.
- האב־טיפוס מאפשר למשתמשים ולמפתחים לשנות את דעתם בשלבים שונים של הפיתוח. בפיתוח מובנה נאלצים לפעמים "להקפיא" את הדרישות לאחר שלב הניתוח ולהמשיך את הפיתוח לאור דרישות אלה בלי להתחשב בשינויים שמתעוררים לאחר מכן. אם בכל אופן מעוניינים ליישם שינויים בדרישות, יש לעצור את הפיתוח,

להסכים על השינויים – דבר שכשלעצמו דורש בדרך כלל משא־ומתן ולעתים רבות ויכוחים בין הלקוח והמפתח – ולתקן את הדרוש תיקון בעקבות זאת. בגישת האב־טיפוס עצם שיטת העבודה היא תהליך של שינוי שמטרתו להגיע להסכמה עד "הרגע האחרון" בנוגע לדרישות האמתיות.

- אב־טיפוס מאפשר להתחיל את בדיקת המערכת ותיקונה בשלבים מוקדמים. ללא אב־טיפוס נאלצים להמתין עם הבדיקות עד שלבי התכנות ומבחני הקבלה של המערכת. אב־טיפוס לעומת זאת נועד להיות אמצעי לבדיקת תקינות התכנון עוד לפני תחילת הקמת המערכת האמתית. כידוע, תיקון טעויות בשלבים מאוחרים יותר של הפיתוח יקר יותר מאשר בשלבים מוקדמים.

בצד היתרונות יש לגישת האב־טיפוס גם חסרונות. אחד החסרונות הוא שהמשתמשים לא תמיד מבינים או מודעים לכך שהתוכנה האמתית עדיין לא פותחה ושנחוצים עוד זמן רב ועלויות עד שזה יקרה.

כבר ציינו שאין הסכמה לגבי המונח אב־טיפוס ויש לו סוגים שונים. כעת נסקור כמה מהם ולאחר מכן נדון ביתרונותיהם ובחסרונותיהם. נציין מראש שיש חפיפה מסוימת בין הסוגים והגבולות ביניהם לא תמיד ברורים.

1.4.3.3 מערכת דמה

מערכת דמה (mock-up) איננה מוצר אמיתי אלא דגם מוקטן שרק דומה למוצר בצורתו החיצונית. מטרתה להראות למנהלים או ללקוחות את צורת המוצר או לבחון כמה היבטים של התכנון. לדוגמה, בתעשיית המכוניות דגם מוקטן של מכונית (מעין צעצוע) יכול לשמש לבחינת הצורה החיצונית (האסתטיקה) וגם למדידת ההשפעה של מהירות הרוח (מבחן נקבת הרוח). בפיתוח תוכנה מערכת דמה מתבטאת בתצוגה על המסך של מנשקים (תפריטים ומסכי קלט ופלט). מרכיבים אלה כמובן אינם יוצרים מערכת תוכנה עובדת אלא רק שלד חיצוני שלה, כפי שהוא נראה למשתמשים. ואולם יש לזכור שבעבור רוב המשתמשים המנשקים מייצגים את המערכת ורק באמצעותם הם מבינים מה היא עושה, ואילו ה"קרביים" שלה (התוכנה עצמה) אינם מעניינם.

אפשר להכין מערכת דמה באמצעות כלי תוכנה שונים, כולל מעבד תמלילים עם יכולת גרפית, תוכנה לעריכת מצגות, מחולל יישומים או כלי CASE ייעודיים. אם משתמשים במעבד תמלילים או בתוכנה לעריכת מצגות, ברור שהדמה יכול לשמש אך ורק לתצוגה אך אי־אפשר להמשיך ולהשתמש בו בהמשך תהליך הפיתוח. לעומת זאת, אם משתמשים

במחולל יישומים או בכלי CASE, אפשר להמשיך ולפתח את הדמה לאב־טיפוס שימושי יותר, כפי שנראה בסעיף הבא.

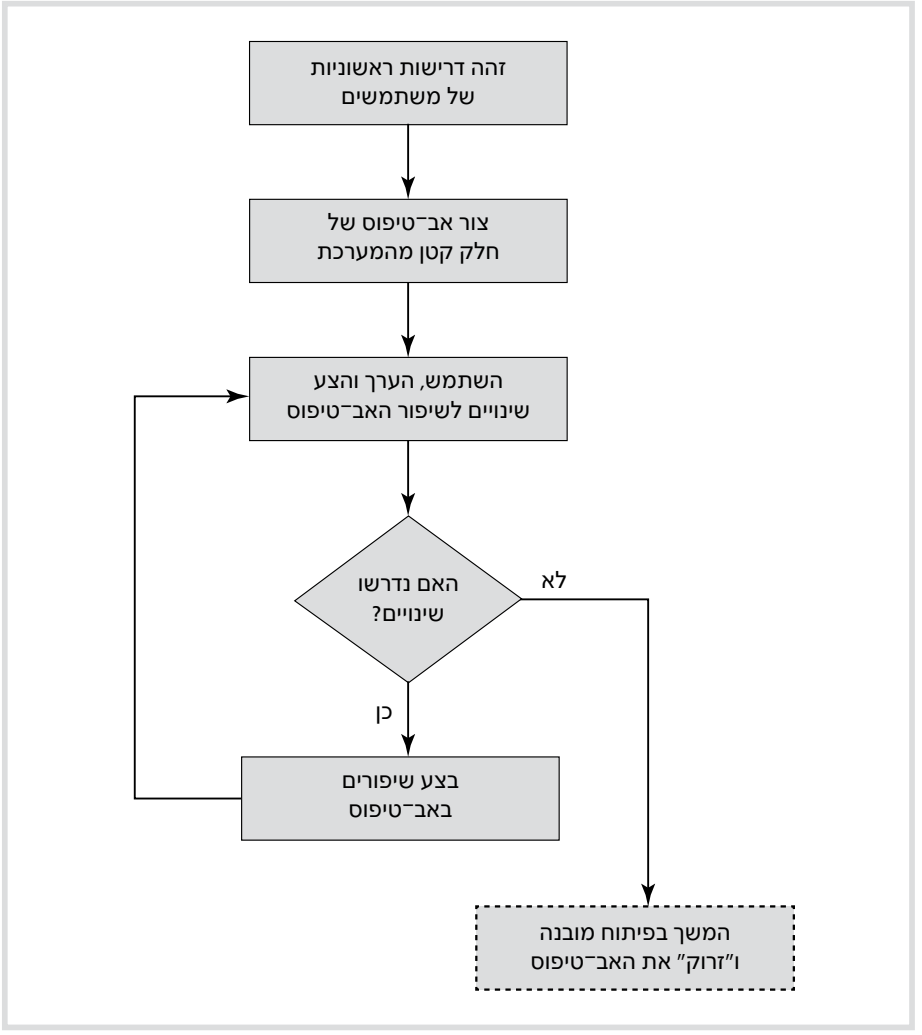
אפשר להכין מערכת דמה בשלבים הראשונים של פיתוח המערכת. לדוגמה, במסגרת הכנת מסמך האפיון הראשוני הצוות המקצועי יכול להכין דמה של חלק מהמערכת הרצויה כהשלמה לאפיונה. דמה כזה יכול לעזור לשכנע את ההנהלה בחיוניותה של המערכת החדשה. דוגמה נוספת: במסגרת הכנת בקשה לקבלת הצעות (RFP) אפשר לדרוש מבתי התוכנה שפונים אליהם לכלול בהצעתם אב־טיפוס של חלק מהמערכת המוצעת. אב־טיפוס כזה יכול להיות דמה כפי שתואר לעיל או אף מערכת תוכנה קטנה מתפקדת שתפותח באמצעות כלי תוכנה מתאים. מטרת אב־טיפוס מעין זה לאפשר לספק פוטנציאלי לשכנע את הלקוח שהוא הבין היטב את הבקשה ולהראות לו (באמצעות המנשקים) איך תיראה המערכת העתידית מנקודת מבטו. איכותו של האב־טיפוס שיכלול כל סָפָק בהצעתו תובא בחשבון בשיקולים של בחירת הספק המועדף.

מובן מאליו שאב־טיפוס מסוג זה אינו בא במקום פיתוח המערכת האמתית, שאותה עדיין צריך לפתח בגישת פיתוח מתאימה.

1.4.3.4 אב־טיפוס שנועד "לזריקה"

אב־טיפוס שנועד "לזריקה" (throw away prototype) הוא מערכת מידע ראשונית, קטנה, שנועדה לשמש להבהרת דרישות ותהליכים ולבחינת תכונות המערכת האמתית. בניגוד למערכת דמה זוהי תוכנה מתפקדת אך היא כוללת רק **חלק קטן** מהמערכת הגדולה והמורכבת שצריך לפתח; היא נועדה להיות זמנית ולסיים את חייה בתום שלב הבחינה שלה. לאחר מכן תיבנה תחתיה ועל־פיה מערכת המידע האמתית. אי־אפשר ולא כדאי להקדיש זמן ומשאבים רבים מדי לבניית מערכת לשימוש זמני בלבד; לכן עובדים בצורה "מהירה וגסה" (quick and dirty). המשמעות היא שלא מקפידים על פיתוח כל מרכיבי המערכת ועל צורתם הסופית או שמתעלמים ממרכיבים ומתכונות מסוימים (כגון יעילות התכניות, עמידה בעומסים עקב שימוש של משתמשים רבים, אבטחה והרשאות גישה).

האב־טיפוס נבנה בשיתוף פעולה בין צוות הפיתוח לנציגי המשתמשים הרלוונטיים. אפשר לפתחו באמצעות מחולל יישומים שעובד במחשב קטן (ואילו המערכת האמתית תפותח לאחר מכן בסביבת חומרה ותוכנה מתאימות). במהלך העבודה נוצרים יחסי עבודה בין המפתחים לנציגי המשתמשים וכל צד נותן ומקבל משוב מִיָדי מהצד האחר. התהליך נמשך עד ששני הצדדים מסכימים שמה שנוצר מתאים לשמש מודל לבניית המערכת האמתית. תרשים 1.10 מדגים את תהליך היצירה של אב־טיפוס שנועד "לזריקה".



תרשים 1.10: תהליך הפיתוח של אב־טיפוס שנועד "לזריקה".

לעומת היתרונות הגלומים בפיתוח אב־טיפוס "לזריקה", יש כמה בעיות הכרוכות ביישומו:

- יש נטייה ליישם באב־טיפוס לזריקה את החלקים הפשוטים של המערכת ולדחות את הטיפול בחלקים המורכבים יותר לשלב הפיתוח של המערכת האמתית. דבר זה נובע מהרצון (המוצדק) לפתח מהר ככל האפשר אב־טיפוס שאפשר להציגו למשתמשים ולקבל מהם משוב. ואולם בכך מחטיאים מטרה אחרת, חשובה לא פחות. התוצאה

עלולה להיות שהחלקים הבעייתיים של המערכת בעצם לא נבדקו ויהיה קשה לעבור מן המודל החלקי והפשוט אל המערכת האמתית המורכבת.

- בגישת האב־טיפוס לזריקה נוצרת תלות בין המפתחים למשתמשים. בעיות ועימותים בין המפתחים למשתמשים עלולים להכשיל את התהליך. לעומת זאת, קשר הדוק מדי עם המשתמשים עלול לגרום להגדרת מערכת שתהיה "צמודה" מדי למצב הקיים בארגון או לרצונותיהם של אותם משתמשים ולא תתחשב בשינויים הנחוצים באופן התפעול של הארגון.
- בניית אב־טיפוס לזריקה היא פרויקט בפני עצמו המצריך מיומנות ניהול גבוהה ופיקוח על תהליך עבודה לא מובנֶה שמתבצע בין המפתחים למשתמשים. לדוגמה, כדי להפיק את מרב התועלת מיצירת האב־טיפוס יש לוודא שהמשתמשים לא יסתפקו במשוב חד־פעמי אלא ימשיכו לתת משוב במחזוריות עד שהאב־טיפוס יענה על הדרישות. גם פעולות אלה צריכות להתבצע לפי תכנון ולוח זמנים. מהירות התהליך והמחזורים הרבים עקב השינויים עד לגיבוש הסופי מקשים על הניהול.
- אין הגדרה מדויקת איך בודקים את האב־טיפוס ומתי יודעים שהוא עונה על הצרכים. לפיכך גם קשה לקבוע מתי האב־טיפוס מסיים את חייו ("נזרק"). קושי גדול עוד יותר הוא לתרגם בצורה יעילה את מה שמצוי ונמצא טוב בָּאב־טיפוס למערכת האמתית שהרי המערכת האמתית תפותח בסביבת פיתוח אחרת.
- נוסף על מגבלות אלה נזכיר שוב שאב־טיפוס "לזריקה" אינו מהווה תחליף לפיתוח מובנֶה. משמע, הוא אינו פותר בעיות מרכזיות בגישת מפל המים – הארכת זמן הפיתוח והגדלת העלויות – שהרי במקביל לעבודה על האב־טיפוס מתבצע התהליך הרגיל של פיתוח כלל המערכת.

1.4.3.5 אב־טיפוס "מתפתח"

אם האב־טיפוס שנועד "לזריקה" נחשב להצלחה, אפשר לשקול לא "לזרוק" אותו אלא להמשיך לפַתחו, כלומר לנצל את תנופת ההצלחה על־ידי הוספת עוד חלקים למערכת תוך שילובם עם החלקים הקודמים עד שתתקבל מערכת המידע האמתית. גישה זו היא למעשה ראשיתה של **גישת הפיתוח המחזורית** (iterative) שעוד ידובר בה בהמשך הפרק.

חשוב להבהיר ש**אב־טיפוס מתפתח** (evolutionary prototype) נוצר על בסיס האב־טיפוס שנועד לזריקה וכהמשכו, כלומר מלכתחילה לא תוכנן לפתח את כלל המערכת בגישה זו. כאן טמון ההבדל הגדול בין גישה זו לגישת "בנה ותקן" שבה הפיתוח של **כלל המערכת** נעשה באופן מתפתח כפי שתואר לעיל. לעומת זאת כאן לא תוכנן מראש לפתח את

כלל המערכת בשיטה זו אלא רק אב־טיפוס של חלק קטן, ורק בגלל הרצון לנצל הצלחה ממשיכים את הפיתוח עד לקבלת מערכת שלמה.

חלק מהבעיות שצוינו בקשר לאב־טיפוס לזריקה תקפות גם לגבי אב־טיפוס מתפתח, ובפרט בעיות שנובעות מהעבודה המשותפת והתלות שבין המפתחים למשתמשים ומקשיי הניהול של התהליך. בעיות נוספות בגישת האב־טיפוס המתפתח הן:

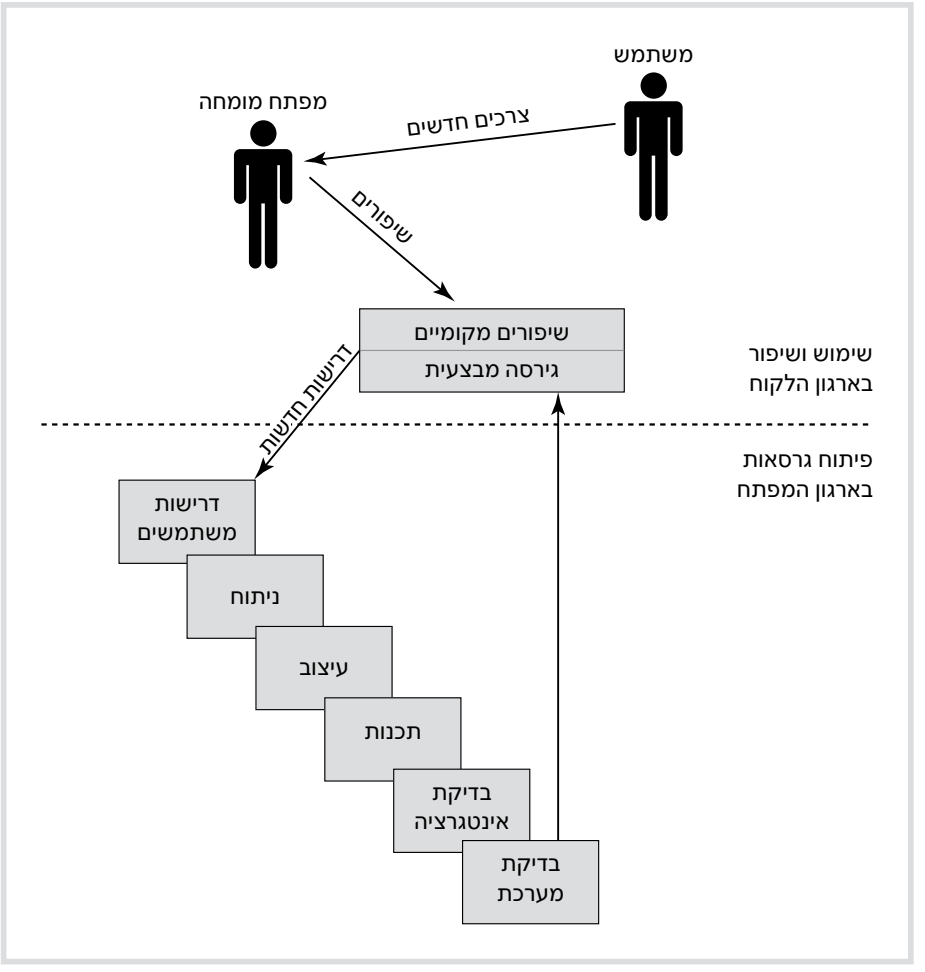
- כשמפתחים את כל המערכת מסתכלים מראש על כל הרכיבים והמורכבות ומתכננים ארכיטקטורה מתאימה שתענה על כל הדרישות, הן הפונקציונליות (הפונקציות שהמערכת צריכה לבצע) והן הלא־פונקציונליות (כגון דרישה לעמוד בעומס). ואולם בפיתוח בחלקים לא רואים את כל התמונה ולכן התוצאה עלולה להיות לא יעילה או לא טובה.
- אי־אפשר להבטיח שהאב־טיפוס הראשון (שנועד לזריקה) יהיה בר־הרחבה לכלל מערכת. נזכור שהוא נבנה מראש בסביבת פיתוח מוגבלת ואילו המערכת האמתית אמורה להיות גדולה ומורכבת בהרבה. ככל שהאב־טיפוס המתפתח מתרחב וכולל עוד ועוד פונקציות של המערכת, כן גדל הסיכון בהוספת חלקים ותכונות חדשות. כמו כן יש בעיה איך לשלב את החלקים החדשים עם החלקים הקיימים במהירות ובסיכון נמוך.
- מכיוון שהוחל בפיתוח המערכת כאב־טיפוס יש נטייה להקפיד פחות על איכותה. הבעיה נובעת מהבנה מוטעית כביכול שמכיוון שהתחילו ביצירת אב־טיפוס גם ההמשך יכול להיעשות באותה דרך, כלומר לא להשקיע יתר על המידה בהגדרת דרישות, ניתוח ועיצוב, תיעוד וכדומה אלא לתכנת במהירות.
- כשהפיתוח של חלק חדש כלשהו מגיע למבוי סתום וצריך לסגת לפיתוח מובנה, יוצא שבוזבזו משאבים וזמן בשל הפיתוח המיותר. נחוצה גישה של "אמצע הדרך" שתפחית את הסיכונים ועם זאת תוביל לאיכות מספקת.

לסיכום, סביר להניח שסיכויי ההצלחה של גישת האב־טיפוס המתפתח קטנים ככל שמערכת המידע שצריך לפתח גדולה ומורכבת יותר.

1.4.3.6 אב־טיפוס מבצעי (אופרטיבי)

אב־טיפוס מבצעי או אופרטיבי (operational prototype) שונה מסוגי האב־טיפוס הקודמים. כאן מדובר בפיתוח מערכת תוכנה אמתית ושלמה שבטרם תוטמע בארגונים עוברת תקופת ניסוי או מבחן המלווה בשינויים ובהתאמות עד לקבלת החלטה שהמערכת

מוכנה. הגישה מתאימה במיוחד לבתי תוכנה המפתחים מערכות גדולות כמו חבילות תוכנה שנועדו להימכר ללקוחות רבים. יישום הגישה מחייב שיתוף פעולה בין המפתח (בית התוכנה) לכמה מלקוחותיו המספקים לו סביבת ניסוי או מבחן למערכת המפותחת. שיטת פיתוח המערכת מתוארת בתרשים 1.11.



תרשים 1.11: תהליך הפיתוח של תוכנה בגישת אב־טיפוס מבצעי.

תהליך הפיתוח מתבצע כך: בתחילה מפתחים מערכת בסיסית הכוללת דרישות ותכונות סטנדרטיות שבית התוכנה מעריך שיתאימו ללקוחות פוטנציאליים רבים. מערכת זו

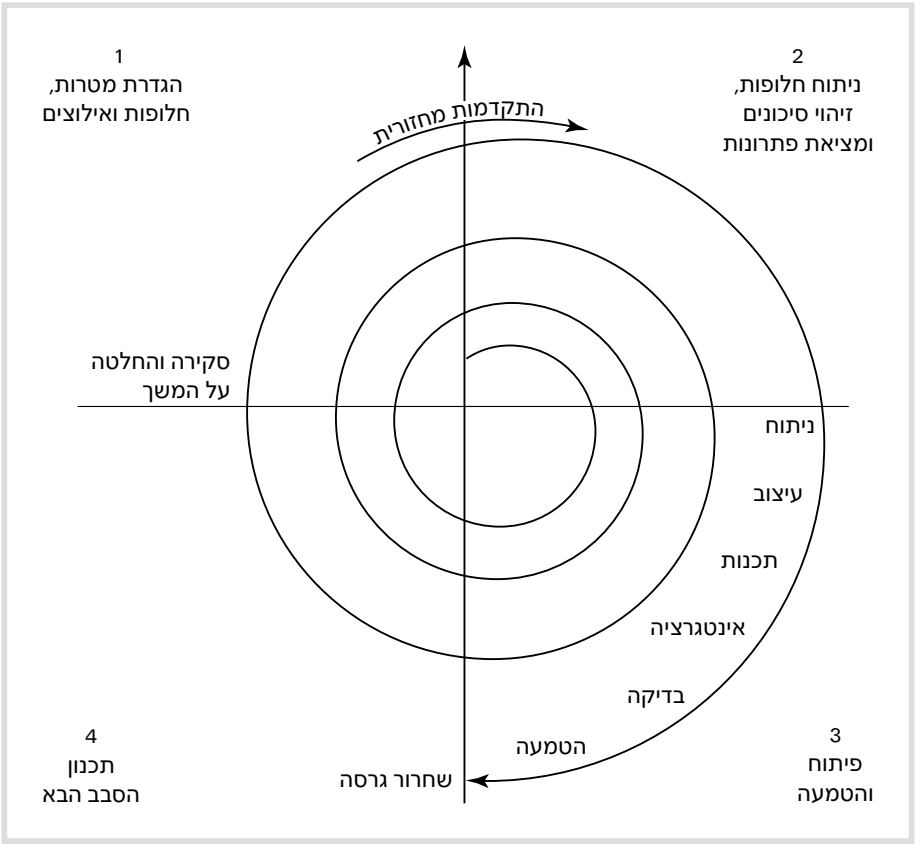
מכונה בשלב זה אב־טיפוס. כעת פורסים מערכת זו אצל הלקוחות המשתתפים בניסוי כדי שיתחילו להשתמש בה. בכל אתר הארגון המפתח מַנֶה מומחה אחד או יותר שיצפה ויעקוב אחר המערכת בפעולה. המשתמשים חושפים בעיות ומעלים לפני המומחה רעיונות ודרישות לתכונות נוספות שהם מצפים מהמערכת. בזמן שהמשתמשים אינם עובדים עם המערכת המומחה מיישם את הדרישות שהועלו ומשלבם בגרסה המתפקדת של המערכת, ואז המשתמשים מתנסים בעבודה אתה ונותנים למומחה משוב שאכן הבין ויישם נכונה את דרישותיהם. השינויים שהוחלט לקבלם באתרי הלקוחות המשתתפים בניסוי מועברים לבית התוכנה ושם הם מוכנסים למערכת ומתקבלת גרסת תוכנה חדשה. את הגרסה החדשה פורסים מחדש באתרי הלקוחות והתהליך יכול לחזור על עצמו בכמה סבבים עד שמתקבלת גרסה סופית שמשביעה את רצון הלקוחות והנהלת בית התוכנה. אם יש לקוחות מסוגים שונים עם דרישות שונות, לא מן הנמנע שבית התוכנה יפתח כמה גרסאות של התוכנה שכל אחת מהן תתאים ללקוחות מסוג אחר.

בין שמכנים פיתוח מערכת בגישה זו אב־טיפוס או בכל שם אחר (כגון גרסאות beta), אין בה התייחסות לגישת הפיתוח של המערכת עצמה בבית התוכנה. סביר להניח שבית תוכנה המעוניין לפתח מערכת איכותית שיוכל למכור בסופו של דבר ללקוחות רבים יישם גישת פיתוח מובנית (כגון מפל המים או גישה עדכנית יותר כפי שיתואר בהמשך).

1.4.4 המודל הספירלי

גישת המודל הספירלי (spiral model) פותחה בסוף שנות ה־80 על־ידי Barry Boehm ואפשר לראותה כגרסה איטרטיבית (iterative, מחזורית) של גישת מפל המים. הגישה פותחה בעבור פרויקטים גדולים מאוד, יקרים וארוכי טווח המשלבים פיתוח של תוכנה וחומרה (כגון פיתוח מערכות צבאיות ומערכות חלל).

ההבדל העיקרי בין הגישה הספירלית לגישת "מפל המים" הוא שפיתוח המערכת בגישה הספירלית אינו נעשה בבת אחת אלא בכמה סבבים. כל סבב יכול להימשך חודשים רבים (בין שישה חודשים לשנתיים) ונחלק לארבעה שלבים: (א) הגדרת מטרות, איתור חלופות והגדרת אילוצים; (ב) ניתוח החלופות, זיהוי סיכונים, מציאת פתרונות לסיכונים ואופציה לבניית אב־טיפוס; (ג) פיתוח חלק מהמערכת כולל תכנון מפורט, תכנות, אינטגרציה עם החלקים הקודמים והטמעה; (ד) תכנון הסבב הבא. בכל סבב פיתוח מופק חלק נוסף של המערכת והוא מסתיים במסירת תוצר ללקוח (release). תרשים 1.12 מציג את המודל.



תרשים 1.12: המודל הספירלי.

הצידוק העיקרי לגישת הפיתוח הספירלי הוא שפיתוח מערכת גדולה, יקרה ומורכבת נעשה בתנאי אי־ודאות ואי־אפשר לאפיין מראש את המערכת לפרטיה – זאת בניגוד להנחה הבסיסית של מודל מפל המים. הדרישות הפונקציונליות, הטכנולוגיה הנדרשת וישימות המערכת בכללותה הולכים ומתבהרים תוך כדי ביצוע הסבבים והעמדת התוצרים שפותחו לבחינה והערכה.

גישה זו שמה דגש ב**ניתוח הסיכונים** (risk analysis) שקיימים בתהליך הפיתוח, הן טכנולוגיים והן ניהוליים, כגון אי־השגת כוח אדם מקצועי או עזיבת אנשי מפתח לפני גמר הפיתוח, פשיטת רגל של יצרן החומרה שהתוכנה מבוססת עליה, השקעה קטנה מדי בבדיקות ובבקרת איכות או פריצת דרך טכנולוגית שבגללה ייעשה הפרויקט כולו חסר ערך. לפיכך בכל סבב, לפני שמתחילים בפיתוח החלק המיועד, בוחנים כל סיכון לעומק

ומציעים לו פתרון. אם אי־אפשר למצוא פתרונות לסיכונים העיקריים ייתכן שיוחלט לעצור את הפרויקט או להמשיך בו בהיקף מצומצם יותר. רק לאחר ניתוח הסיכונים עוברים לפיתוח החלק שתוכנן באותו סבב.

ואולם קיימות גם מגבלות ביישום המודל. מגבלה אחת נובעת מהחשש מפני תביעות משפטיות. אם מדובר בפיתוח פרויקט ביחידת הפיתוח של הארגון עצמו, אפשר להתגבר ביתר קלות על בעיות הנובעות מהחלטה להפסיק את הפרויקט, למשל על־ידי שילוב צוות הפיתוח בפרויקט אחר. ואולם אם פיתוח הפרויקט נעשה בידי חברה חיצונית, ההחלטה להפסיקו עלולה לגרור תביעה משפטית.

מגבלה נוספת של המודל הספירלי נובעת מהיקף הפרויקט. המודל מתאים רק לפרויקטים גדולים. אין טעם לבצע בדיקת סיכונים בפרויקטים קטנים שעלות הבדיקה בהם יכולה להיות גבוהה כעלות הפרויקט כולו.

היתרון העיקרי של המודל – ניתוח סיכונים – הוא גם חסרונו. אם המפתחים אינם מיומנים באיתור מוקדי סיכון אפשריים ובניתוח מדויק של סיכונים, יש חשש שצוות הפיתוח ירגיש בטוח במלאכתו אף־על־פי שלאמיתו של דבר הפרויקט נמצא במצב גרוע. רק אם חברי צוות הפיתוח מוכשרים לנתח את הסיכונים כדאי להנהלה לבחור במודל זה לפיתוח המערכת.

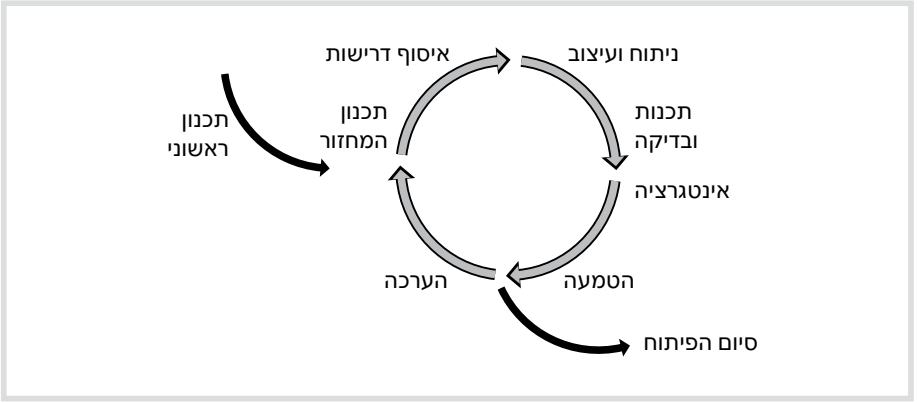
1.4.5 פיתוח תוספתי־מחזורי

גישת הפיתוח התוספתי־מחזורי (Incremental Iterative Development, **IID**) כללית יותר מגישת המודל הספירלי. הגישה הספירלית נועדה בעיקר בעבור מערכות מורכבות, יקרות וארוכות טווח המשלבות פיתוח תוכנה וחומרה. לעומת זאת המוטו של גישת IID הוא ש**כל** מערכת מידע גדולה צריכה להיבנות באופן מודולרי, נדבך על נדבך, בין בשל אילוצים כמו מגבלת תקציב ומחסור בכוח אדם מקצועי ובין מפני שאי־אפשר ללמוד ולהגדיר מראש ובצורה מדויקת את כל הצרכים אלא רק תוך כדי התקדמות בעבודת הפיתוח. לכן יש לפתח מערכת בצורה מחזורית תוך הוספת חלקים חדשים לחלקים קיימים עד שלבסוף מתקבלת המערכת השלמה שעונה על כל הדרישות.

חשוב לציין שגישת IID לא באה להחליף את גישת המודל הספירלי; בעצם היא הייתה קיימת לפניה והתפתחה במשך שנים רבות במקביל לגישת מפל המים. מאז שנות ה־70 היא יושמה בארגונים רבים, בגרסאות רבות ובשמות שונים. במילים אחרות, גם בשנים שרבים נהו אחרי גישת מפל המים היו אנשי מקצוע, חוקרים וארגוני פיתוח שטענו

שאי־אפשר לפתח בהצלחה מערכת גדולה בבת אחת, באופן סדרתי, אלא צריך להתחיל בפיתוח של חלק מהדרישות ולהמשיך את הפיתוח באופן תוֹסַפתי (incremental).

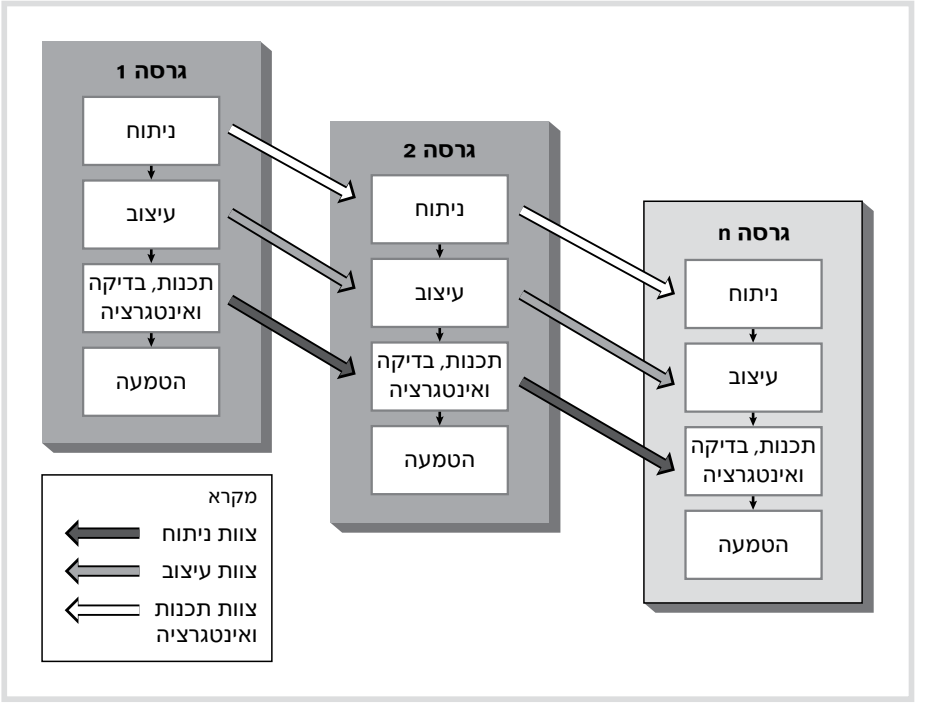
כאמור, לגישה זו יש גרסאות רבות ואפשר להדגים אותה בצורות שונות. שתי גרסאות מודגמות בתרשימים 1.13 ו־1.14. תרשים 1.13 מדגים את המחזוריות בפיתוח באמצעות מעגל; הכניסה למעגל חלה לאחר שמתבצע שלב תכנון ראשוני; המעגל עצמו מציין מחזורי פיתוח והיציאה ממנו מתרחשת כשמסתיים פיתוח הנדבך האחרון של המערכת. תרשים 1.14 מדגים את הגישה בצורה "פרוסה", היינו כסדרת פיתוחים.



תרשים 1.13: פיתוח תוספתי־מחזורי – גרסת המעגל.

כפי שמראה תרשים 1.13, בתחילת הפרויקט יש שלב תכנון ראשוני. אפשר להניח שבשלב זה מבצעים אפיון ראשוני של כלל המערכת, מחליטים מהן מערכות המשנה ומה יהיה סדר פיתוחן, מהי מסגרת התקציב ומהו לוח הזמנים. כמו כן מחליטים על סביבת הפיתוח, ארכיטקטורת המערכת וכדומה. הפיתוח עצמו נעשה באופן מחזורי; בכל מחזור מפתחים חלק או מערכת משנה מסוימת, ובתום המחזור משלבים את החלק החדש עם החלקים הקודמים. אפשר להניח שכל שלב במחזור הפיתוח מבוצע בשיטות ובטכניקות עבודה איכותיות ושמתעדים את תוצרי הפיתוח. אפשר לומר אפוא שבכל מחזור פיתוח מיישמים "מפל מים זוטא".

אפשר לבצע את מחזורי הפיתוח בצורות שונות. אפשרות אחת היא שצוות פיתוח אחד עובד על פיתוח של חלק מסוים (מערכת משנה) ולאחר שסיים לפתח אותו עובר לחלק הבא, וכך הלאה. בשיטה זו לא נדרש כוח אדם רב אך משך הפיתוח הכולל ארוך יחסית. אפשרות אחרת היא שכמה צוותים עובדים במקביל, כל צוות עובד על חלק מסוים ולאחר



תרשים 1.14: פיתוח תוספתי־מחזורי – גרסת הפרוסות.

שסיימו משלבים את החלקים. בשיטה זו נדרשים כמובן יותר משאבי אנוש אך משך פיתוח המערכת קצר יותר. שיטת ביניים המודגמת בתרשים 1.14 היא עבודה במדורג. בשיטה זו יש גם חלוקת עבודה פנימית: כשמנתחי המערכת מסיימים את עבודתם על חלק מסוים הם עוברים לנתח את החלק הבא של המערכת; במקביל נמשך פיתוח החלק הקודם על־ידי מעצבים ומתכנתים שגם הם עובדים באופן מדורג.

למעשה כבר ראינו גישות שיישמו את הרעיון של פיתוח בסבבים (מלבד המודל הספירלי), דהיינו: גישת "בנה ותקן" וגישת האב־טיפוס המתפתח, אבל יש הבדלים מובהקים ביניהם לגישת הפיתוח התוספתי־מחזורי. בגישת "בנה ותקן" אין תכנון, אין הגדרת דרישות ראויה, אין הבחנה בין שלבים, אין ניהול ובקרה, לא מיישמים שיטות וטכניקות בשלבי הפיתוח השונים; הדגש מושם בתכנות כמעט מיָדי ומשלא מתקבלת תוצאה טובה (כצפוי) נאלצים לתקן ולחזור ולתקן. בגישת האב־טיפוס המתפתח ראינו שמלכתחילה מתכוונים רק לפתח אב־טיפוס "לזריקה" של חלק קטן מהמערכת, ורק אם מתאפשר מנצלים את ההצלחה ומנסים להוסיף לו עוד חלקים, וכך נוצרת מעין מחזוריות בתהליך

הפיתוח. ואילו בגישת הפיתוח התוספתי־מחזורי, כמו גם בגישת המודל הספירלי, התפיסה היא שמלכתחילה מתכננים לבנות את המערכת בחלקים, מחליטים מראש מה יהיו החלקים ובאיזה סדר יפותחו. כמו כן בונים את המערכת מלכתחילה באמצעות כלי הפיתוח וסביבת הפיתוח ה"אמתיים" שלה, משלבים כל חלק חדש עם החלקים הקודמים ומטמיעים את החלקים שפותחו בארגון.

אחד הקשיים בגישת הפיתוח התוספתי־מחזורי הוא שכל חלק חדש שמפותח צריך להשתלב איכשהו בחלקים הקיימים מבלי לפגוע בהם, והמערכת שהוקמה עד אז צריכה להתאים את עצמה להרחבה. קושי נוסף נובע מהחשש שהפיתוח עלול להידרדר לגישת "בנה ותקן", בפרט אם התהליך אינו מתוכנן מראש ואם אין מקפידים על שימוש בשיטות ובטכניקות מתאימות במהלך הפיתוח של כל חלק. אם המערכת מורכבת ממספר רב מדי של מערכות משנה, יידרש זמן רב לביצוע השילוב בכל שלב. כמו כן מנהל הפיתוח עלול לאבד שליטה ובקרה על התהליך כולו.

1.4.6 סיכום הגישות לפיתוח מערכות מידע

בפרק זה הצגנו כמה גישות לפיתוח מערכות מידע ותוכנה, תוך הדגשת יתרונותיהן וחסרונותיהן.

- גישת "בנה ותקן" אינה גישה מובנית ואי־אפשר להסתמך עליה בפיתוח מערכות מידע גדולות ומורכבות, אם כי אין מניעה להשתמש בה בפיתוח מערכות קטנות שאינן קריטיות לארגון.
- גישת "מפל המים" הסדרתית הייתה נפוצה מאוד מאז שנות ה־70 של המאה ה־20. יתרונה בהדגשת הצורך לתכנן, לבצע ולתעד את תוצרי כל אחד משלבי הפיתוח ולאמת אותם בטרם עוברים לביצוע השלב הבא. חסרונותיה העיקריים הם שהיא מאריכה את זמן הפיתוח ולפיכך גם את עלות הפרויקט; ובפרט, היא מחייבת הגדרה מראש של כל הדרישות – משימה כמעט בלתי אפשרית במערכות גדולות ומורכבות – ומתקשה להתמודד עם שינויים בדרישות הנובעים מעצם משך הפיתוח הארוך.
- גישת האב־טיפוס עונה במידת מה על חלק מהבעיות מכיוון שהיא מאפשרת להראות ללקוח כבר בשלב מוקדם של הפיתוח חלק קטן של המערכת, לקבל משוב ולפתח את המערכת האמתית בהתאם. ואולם גישה זו איננה תחליף לגישת מפל המים ואינה מתגברת על רוב מגבלותיה אלא אם כן אפשר לנצל הצלחה ולפתח את האב־טיפוס שנועד מלכתחילה לזריקה לכדי מערכת שלמה. ואולם אפשרות זו אינה ישימה במקרים רבים.

- חלופה אחרת לגישת מפל המים היא גישת המודל הספירלי שמיועדת בפרט למערכות גדולות, יקרות במיוחד ומורכבות המשלבות פיתוח תוכנה וחומרה. גישה זו משלבת עקרונות של פיתוח מובנֶה כמו בגישת מפל המים עם פיתוח מחזורי של המערכת תוך הדגשת היבטי הניהול של הפרויקט המורכב, ובכלל זה בחינת חלופות, ניתוח סיכונים, בקרה ומשוב.
- הגישה התוספתית־מחזורית, שאפשר לראותה כשילוב של גישות מחזוריות שונות, נחשבת כיום לגישה הנכונה לפיתוח מערכות מידע וכתחליף הולם לגישת מפל המים הסדרתית (אלא אם כן המערכת קטנה ופשוטה ואפשר לפתחה בבת אחת ולא בצורה מחזורית). בעקבות התרחבות השימוש בגישה זו במשך השנים היא התפתחה עוד יותר בשנים האחרונות וקיבלה שם חדש – הגישה ה"זריזה" (agile development). בגלל חשיבותה ותפוצתה הרחבה בימינו נפרט אותה בפרק 5, האחרון ביחידה זו.

שאלות חזרה ושינון

1. מתי מתאים ומתי לא מתאים לפתח מערכת בגישת "בנה ותקן"? (ראו סעיף 1.4.1.)
2. מה גורם לנטייה של מפתחים לעבוד בגישת "בנה ותקן"? (ראו סעיף 1.4.1.)
3. מהם המאפיינים העיקריים של מודל "מפל המים? האם מודל זה הוא אמנם גישת פיתוח סדרתית בלבד? (ראו סעיף 1.4.2.)
4. מהן המגבלות והבעיות שנגרמות בגלל פיתוח בגישת "מפל המים"? (ראו סעיף 1.4.2.)
5. מה ההבדל בין אב־טיפוס בחומרה ואב־טיפוס בתוכנה? (ראו סעיף 1.4.3.1.)
6. על אילו מגבלות במודל "מפל המים" עונה גישת האב־טיפוס? (ראו סעיף 1.4.3.2.)
7. מהי מערכת דמה, באילו שלבי פיתוח היא משמשת ובאילו אמצעים אפשר ליצור אותה? (ראו סעיף 1.4.3.3.)
8. מהו אב־טיפוס "לזריקה" וכיצד משתלב פיתוחו בתהליך פיתוח מובנה? (ראו סעיף 1.4.3.3.)
9. אילו בעיות עלולות להתעורר כשמפתחים אב־טיפוס לזריקה? (ראו סעיף 1.4.3.3.)
10. מהו אב־טיפוס מתפתח? מה ההבדל בין פיתוח מערכת בגישה זו לעומת פיתוח בגישת "בנה ותקן"? מה ההבדל בין אב־טיפוס מתפתח לאב־טיפוס לזריקה? (ראו סעיף 1.4.3.3.)
11. אילו בעיות עלולות להתעורר בגישת האב־טיפוס המתפתח? (ראו סעיף 1.4.3.3.)
12. מהו אב־טיפוס מבצעי, כיצד בונים אותו ומה הקשר בינו לבין אב־טיפוס מתפתח ופיתוח בגישת מפל המים? (ראו סעיף 1.4.3.3.)
13. בעבור אילו סוגי מערכות מתאים פיתוח לפי המודל הספירלי? (ראו סעיף 1.4.4.)
14. תארו את שלבי המודל הספירלי ומה נעשה בכל שלב. (ראו סעיף 1.4.4.)
15. תארו את שלבי הפיתוח לפי גישת הפיתוח התוספתי־מחזורי (IID). (ראו סעיף 1.4.5.)
16. השוו בין גישת IID לגישת "בנה ותקן", למודל הספירלי ולגישת האב־טיפוס המתפתח. (ראו סעיף 1.4.5.)

שאלות לעבודה עצמית

1. בחרו בית תוכנה או ארגון שמפתח בעצמו מערכות מידע. ראיינו את מנהלי הפיתוח של כמה פרויקטים שפותחו בארגון בשנים האחרונות. כתבו דוח שיתייחס לנושאים האלה בקשר לגישות הפיתוח:
   א. באיזו גישת פיתוח עבדו בכל אחד מהפרויקטים הנסקרים?
   ב. מי החליט על גישת הפיתוח ומה היו השיקולים (כגון מאפייני הפרויקטים)?
   ג. האם בדיעבד נוהל הפיתוח הִתאים או לא התאים לפרויקט? נמקו.

2. בחרו בית תוכנה (רצוי גדול) או ארגון המפתח מערכות מידע ובצעו בו סקר באמצעות שאלון שיופנה למנהלי הפיתוח במטרה להכין דוח על גישות הפיתוח שבהם השתמשו בפרויקטים של השנים האחרונות. הדוח יתייחס לנושאים האלה:
   א. באיזו גישת פיתוח השתמשו בכל אחד מהפרויקטים? מה הייתה מתודולוגיית הפיתוח? מה היו מאפייני הפרויקטים? מה היו השיקולים לבחירת גישת הפיתוח?
   ב. אילו לקחים הופקו מגישת הפיתוח בכל פרויקט? האם מנהלי הפיתוח שבעי רצון מגישת הפיתוח שיישמו בכל אחד מהפרויקטים? נמקו.