

אלמנטים של תהליכי פיתוח תוכנה



מירב שקרון, meravgu@gmail.com,
יוסי זגורי, yossiza@ariel.ac.il, 052-4668866

אג'נדה

הנדסת דרישות 

תהליכי עבודה 



הנדסת דרישות

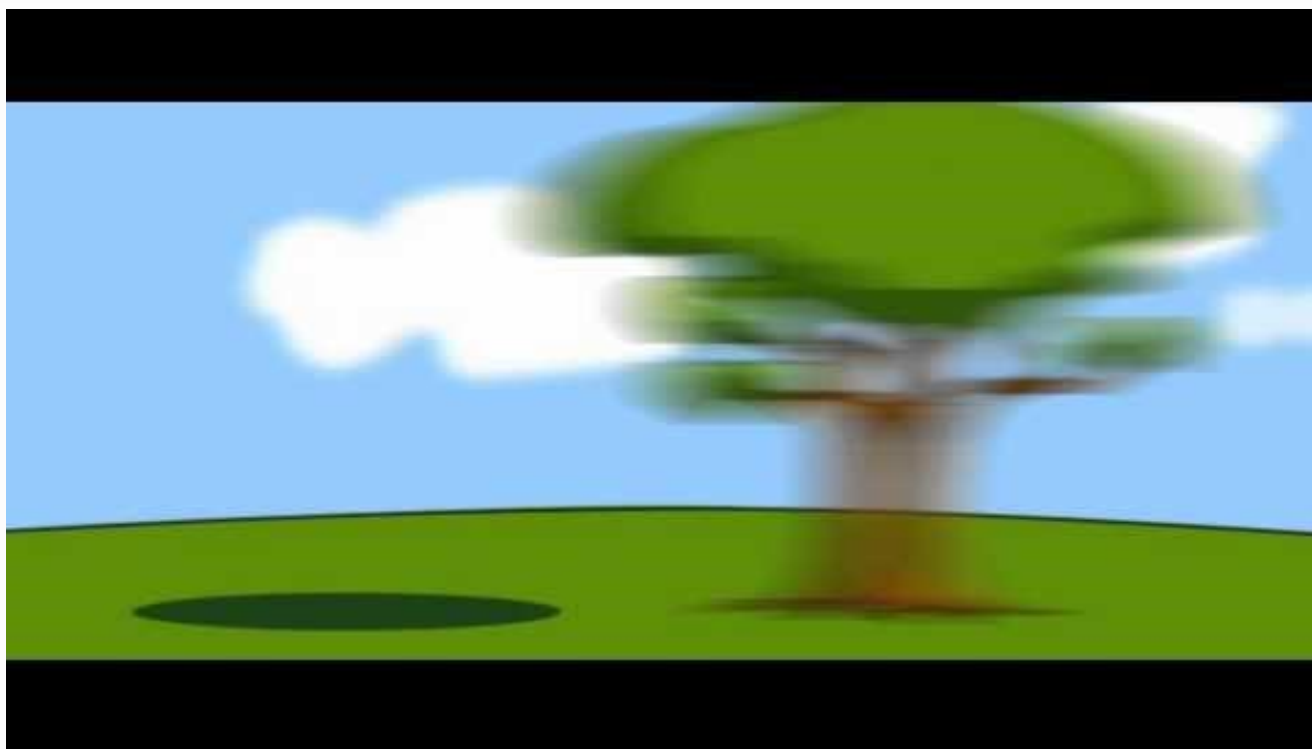
שלב הגדרת דרישות

- מטרת שלב הדרישות הוא יצירת תשתית למפרט התוכנה על פי צורכי הלקוח
 - הגדרת צורכי הלקוח
 - הגדרת יכולות המוצר
 - התנאים בהם המוצר נדרש לעמוד
- הוא הבסיס להבנה משותפת בין הלקוח למפתח.

חשיבות הגדרת דרישות

- מקנה סיכוי גבוה יותר שהתוצרים יענו על דרישות הלקוח.
- מקדים את תהליכי האפיון והפיתוח של המערכת.
- משפר את היכולת לבצע שינויים במהירות תוך בקרה ובחינת השפעתם על דרישות אחרות.
- תיאום ציפיות בין ספק ללקוח על בסיס מנותח, מוסכם ומאושר.
- שיפור היכולת לבצע בקרה על התקדמות הפרויקט.
- שיפור היכולת לביצוע אמידת עלויות ע"י ניתוח של עלות תועלת עבור כל דרישה בנפרד או עבור מכלול דרישות.

דרישות ובעלי עניין



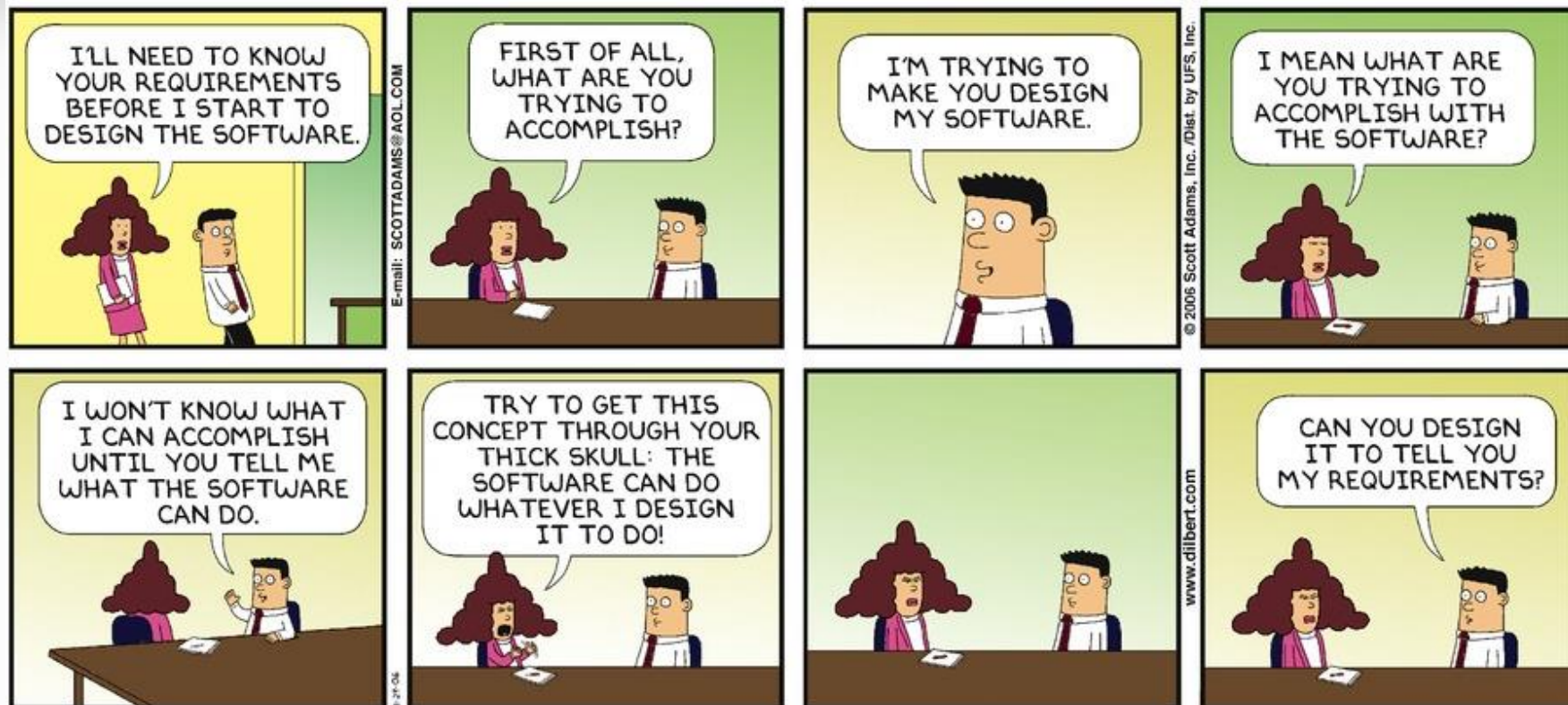
כל בעל עניין "רואה" את המערכת אחרת בעיני רוחו ולכן
חשוב להגדיר מסמך דרישות ברור!

הגדרת דרישה

- תנאי או יכולת הדרושים על ידי בעלי העניין כדי לפתור בעיה או להשיג מטרה.
- תנאי או יכולת שיש למלא או למצוא פתרון כדי לספק התחייבות, תקן, מפרט או מסמכים רשמיים אחרים.
- דרישות אמורות לתאר "מה" המערכת אמורה לעשות ולא "איך" המערכת אמורה לעבוד

שיטות למציאת הדרישות

- ראיונות ושאלונים למשתמשים השונים
- סיעור מוחות
- יצירת demo
- למידה ממערכת קיימת / למידה ממצב קיים



בעיות אופייניות לשלב הגדרת הדרישות

המפתחים חושבים
שהם יודעים מה
המשתמשים רוצים

המנתחים מניחים מה
המשתמשים רוצים

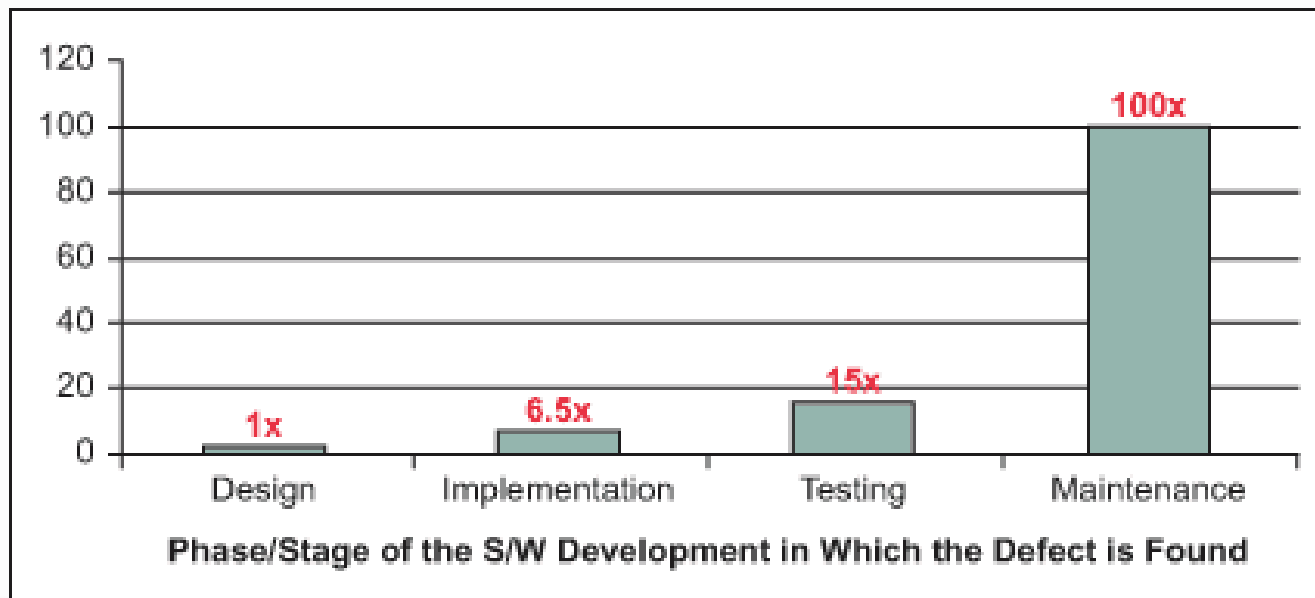
המשתמשים חושבים
שהם יודעים מה הם
רוצים, עד שהם רואים
את התוצאה בעיניים

המשתמשים לא
יודעים להסביר מה
הם רוצים - IKIWISI

אין אמון בין בעלי
העניין

עלות תיקון שגיאות בשלבים השונים- עקומת בוהם

- “bugs are always more expensive to fix later on in the process”
- Boehm, Barry W., John R. Brown, and Mlity Lipow. "Quantitative evaluation of software quality." *Proceedings of the 2nd international conference on Software engineering*. IEEE Computer Society Press, 1976.



הניסוי של IBM

Briski, K. A., et al. "Minimizing code defects to improve software quality and lower development costs." *Development Solutions. IBM. Crawford, B., Soto, R., de la Barra, CL* (2008).



Design and architecture	Implementation	Integration testing	Customer beta test	Postproduct release
1X*	5X	10X	15X	30X

*X is a normalized unit of cost and can be expressed in terms of person-hours, dollars, etc.

Source: National Institute of Standards and Technology (NIST)†

עודף דרישות

- יש לשים לב בשלב זה להגדיר את מה שצריך ולא להגדיר את מה שלא צריך
- כל דרישה משמעה עלויות כספיות, סיבוכיות למערכת וניהול של הבאגים- לא נרצה לעשות את זה במקרה של דרישות אזוטריות או לא שימושיות

סוגי דרישות

- דרישות פונקציונליות (עסקיות): **Functional Requirements (FR)**
מה המערכת אמורה לעשות/להגיב מנקודת המבט של המשתמש
לדוגמא:

- פונקציות
- שירותים

- דרישות לא פונקציונליות (טכניות/ איכות הפתרון) - **Non-Functional Requirements (NFR)**

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

- זמני תגובה/ביצוע
- נפחי פעילות
- אמינות
- אבטחת מידע
- אופני מימוש
- תדירות ביצוע
- עמידה בעומסים
- שימושיות

דרישות פונקציונליות

- דרישה תפעולית

– (OR = Operational Requirement)

– דרישה המתייחסת לתפעול, לאינטראקציה או להתנהגות של המוצר

- דרישת מידע

– (DR = Data Requirement)

– דרישה המתייחסת לישויות המידע ולנתונים בהן נדרשת התוכנה לטפל (לקלוט, לאחסן, לאחזר, לעבד, להפיק כפלט)

– לדוגמא: נתונים ומבני נתונים, מאגרי מידע/ בסיסי נתונים, דרישות קלט/פלט

דרישות לא פונקציונליות

איכות הפתרון

אילוצים

QA =) מאפייני איכות
(Quality Attributes

PR =) דרישות ביצועים
= Performance
(Requirements

HC =) אילוץ חומרה
Hardware
(Constraint

IC =) אילוץ מימוש
Implementation
(constraint

MC =) אילוץ ניהולי
Management
(Constraint

- reliability - אמינות
- availability - זמינות
- security - ביטחון
- maintainability - אחזקתיות
- usability - שימושיות

- זמן תגובה
- נפח אחסון
- ניצולת מעבד
-

- רכיבים
- ארכיטקטורה
-

- אלגוריתם
- מבנה נתונים
-

- תקציב
- ל"ז
- זמינות משאבים
- התאמה לתקן
-

דוגמא לדרישות פונקציונליות

תפעולית

– המערכת תאפשר להזין הזמנות מלקוח למוצרים שבמלאי.
המערכת תייצר מספר הזמנה חד ערכי בעת שמירת
ההזמנה.

מידע

תפעולית

– כל הזמנה תאפשר לציין למוצר יחידת מידה וכמות. כל שינוי
ביחידת מידה מחייב רישום כפריט נפרד בהזמנה.

מידע

– לכל פריט בהזמנה יש לרשום יחידת מידה, כמות ומחיר
ליחידת מידה.

תפעולית

– ניתן להזמין מוצרים הן דרך מרכז ההזמנות והן בצורה ישירה
באינטרנט

– עבור כל פריט שיוצג יש להציג את שמו, הברקוד שלו וצבעו.

מידע

דוגמא לדרישות לא פונקציונליות

מאפייני
איכות

אילוצ
ניהולי

אילוצ
חומרה

- המערכת תתבסס על מחשב מסוג
- המערכת תהיה זמינה ** שעות ביממה
- עלויות הפיתוח לא יהיו גבוהות מ- \$M200
- תאריך היעד של המערכת הוא 01/01/2018
- יש להשתמש בחישוב הפרמיה החודשית כפי שמחושב במערכת הקיימת
- יש להחזיר תשובה לחיפוש תוך 2 שניות לכל היותר

אילוצ
ניהולי











אילוצ
מימוש

דרישות
ביצועים

מהי דרישה איכותית?

- Identified - בדידה , מזוהה חד ערכית , שייכות ברורה
- Understandable – מובנת (ברורה, מדויקת) - מנוסחת בשפת הלקוח
- Unambiguous - לא עמומה (חד משמעית)
- Complete – שלמה
- Necessary - הכרחית- בעלת תרומה משמעותית לשיפור תהליכי העבודה
- Consistent – עקבית (לא סותרת דרישות אחרות)
- Verifiable - ניתנת לבדיקה באמצעות מבחני קבלה
- Traceable - עקיבה (גם לדרישות ברמה גבוהה יותר וגם בהמשך האפיון)
- Prioritized - מתועדפת

איכותית או לא?

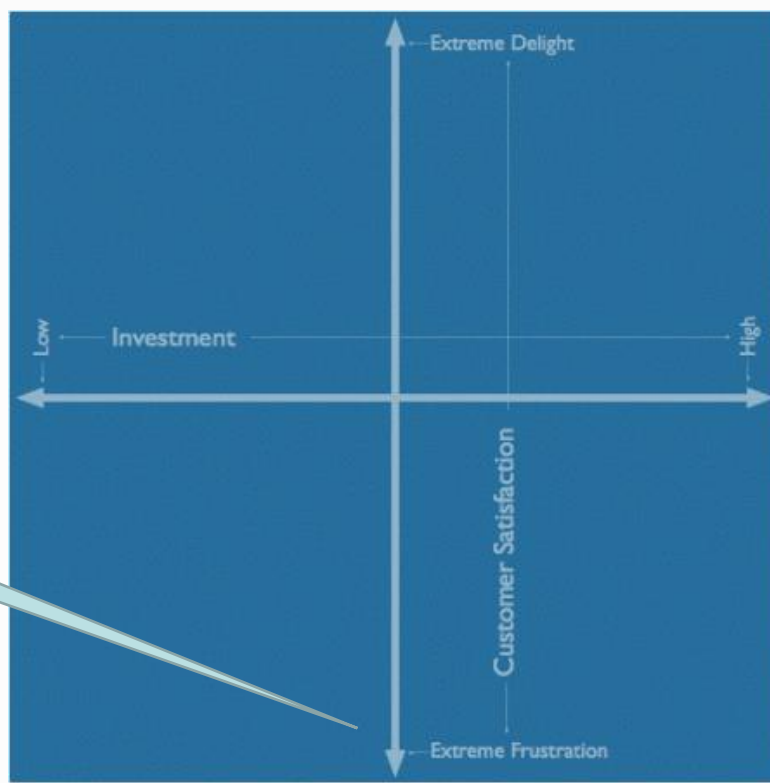
-  חיפוש מתקדם שבו ניתן יהיה לסנן גם לפי טקסט חופשי
-  זמן חיפוש סביר
-  המערכת תפחית את כמות הניירת בעלות שנתיית של כ 100,23 ₪
-  למנהל האתר תהיה האופציה להוציא דוחות לפי חתכים שונים
- במקרה שהמעלית נתקעה במהלך נסיעה מזעיק הנוסע חילוץ באמצעות כפתור החילוץ 
-  המערכת תאפשר חלוקת עבודה מאוזנת והוגנת בין העובדים
-  סטטיסטיקות
-  צמצום הוצאות החברה ב 27 מיליון ש"ח עד מחצית הראשונה לשנת 2018
-  הצגת היסטוריה של תלמיד בכניסה לאתר
-  ידידותי למשתמש

Identified - בדידה , מזוהה חד ערכית , שייכות ברורה
 Understandable – מובנת (ברורה, מדויקת)- מנוסחת בשפת הלקוח
 Necessary - הכרחית- בעלת תרומה משמעותית לשיפור תהליכי העבודה
 Complete - שלמה
 Consistent – עקבית (לא סותרת דרישות אחרות)
 Unambiguous - לא עמומה (חד משמעית)
 Verifiable - ניתנת לבדיקה באמצעות מבחני קבלה
 Traceable - עקיבה (גם לדרישות ברמה גבוהה יותר וגם בהמשך האפיון)
 Prioritized - מתועדפת

מודל Kano

- הוגדר על ידי Noriaki Kano
- Kano, Noriaki. "Attractive quality and must-be quality." *Hinshitsu (Quality, The Journal of Japanese Society for Quality Control)* 14 (1984): 39-48.
- קיבל את פרס Demming לאיכות ב- 1997
- קרא תיגר על התפיסה שרמת שביעות הרצון של לקוחות מבוססת על "כל המרבה הרי זה משובח"
- בעיני הלקוחות יש הבדלים מהותיים בין התכונות השונות של מוצר שתורמות לשביעות רצון.
- המודל מגדיר את הקשר בין רמת ההשקעה בדרישה לעומת שביעות רצון הלקוחות.
- המודל משמש לתיעדוף דרישות במערכת

מודל Kano - הצירים

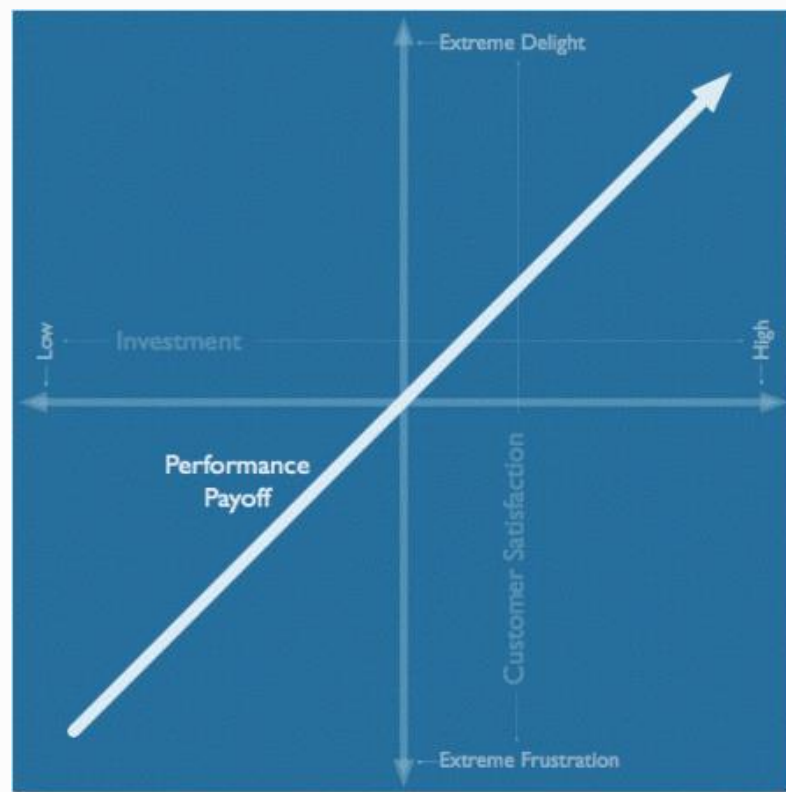


ציר y – שביעות רצון

ציר x – השקעה

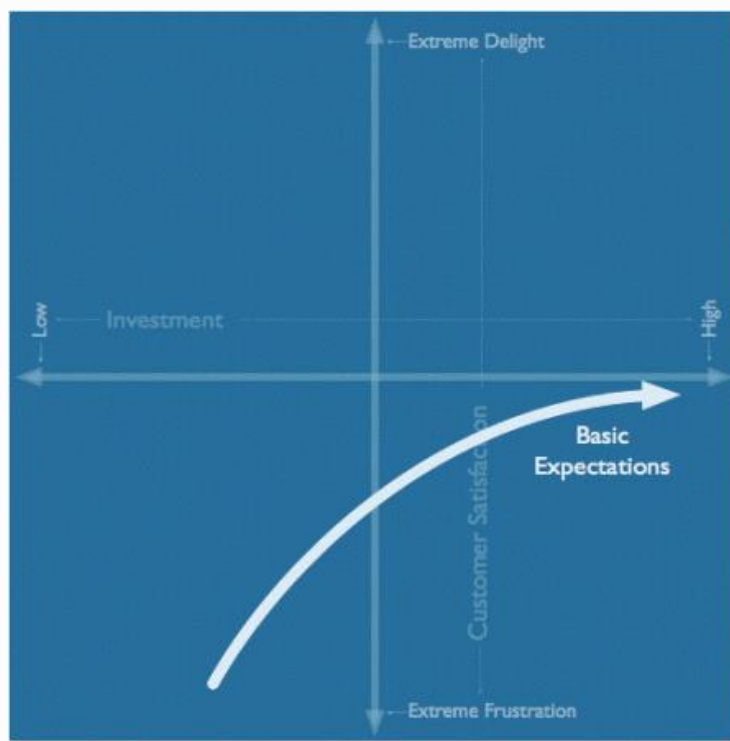
<http://uxtasy.com/blog/2011/01/kano-mode/>

מודל Kano – השתלמות הביצוע



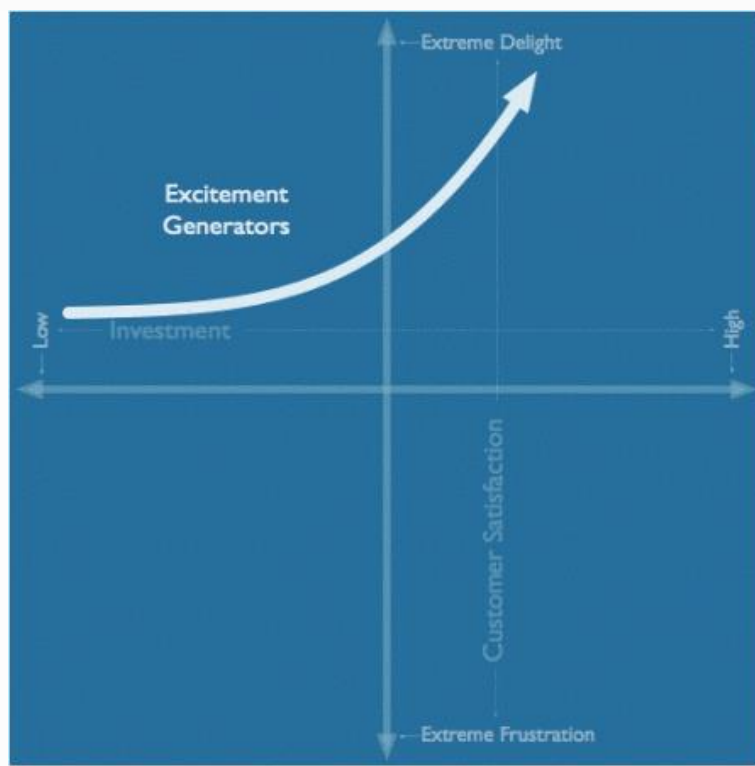
<http://uxtasy.com/blog/2011/01/kano-mode/>

מודל Kano – ציפיות בסיסיות



<http://uxtasy.com/blog/2011/01/kano-mode/>

מודל Kano – מייצרי הנאה



<http://uxtasy.com/blog/2011/01/kano-mode/>



שיטות עבודה

פעילויות בפיתוח התוכנה

- **ייזום** - זיהוי בעיות והזדמנויות
- **הגדרת דרישות** – תיאור מדויק של הנדרש
- **ניתוח** - כיצד ניתן לפתור את הבעיות
- **עיצוב** - בחירה ותכנון הפתרון המתאים
- **מימוש** - תרגום התוכניות למציאות
- **בדיקות** - בחינת התוצאות מול התכנון
- **שילוב** – התקנת המערכת והטמעה אצל המשתמשים
- **תחזוקה** - תהליך מתמשך של ניפוי שגיאות והרחבות

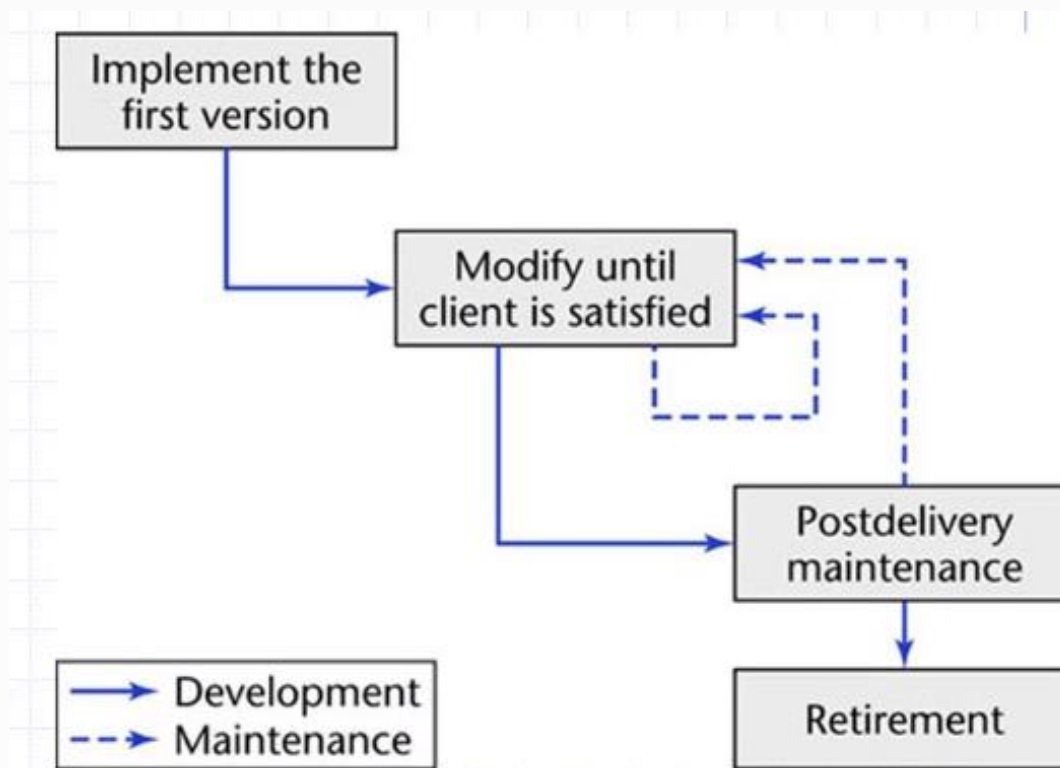
דיונים, הערכת סיכונים, תיעוד, תיעדוף, ארכיטקטורה....

Software Development Life Cycle

- תהליך בניית מוצר תוכנה
- מספר מודלים אלטרנטיביים
- כל מודל מתאר גישה ותהליך = אוסף שלבים
האמורים להתבצע במסגרת הפיתוח
- למודלים השונים מטרה זהה: התמודדות יעילה
עם אי ודאות ועמידה ביעדי עלות ותועלת

SDLC != PCL (Project Life Cycle)

Code and Fix





Code and Fix

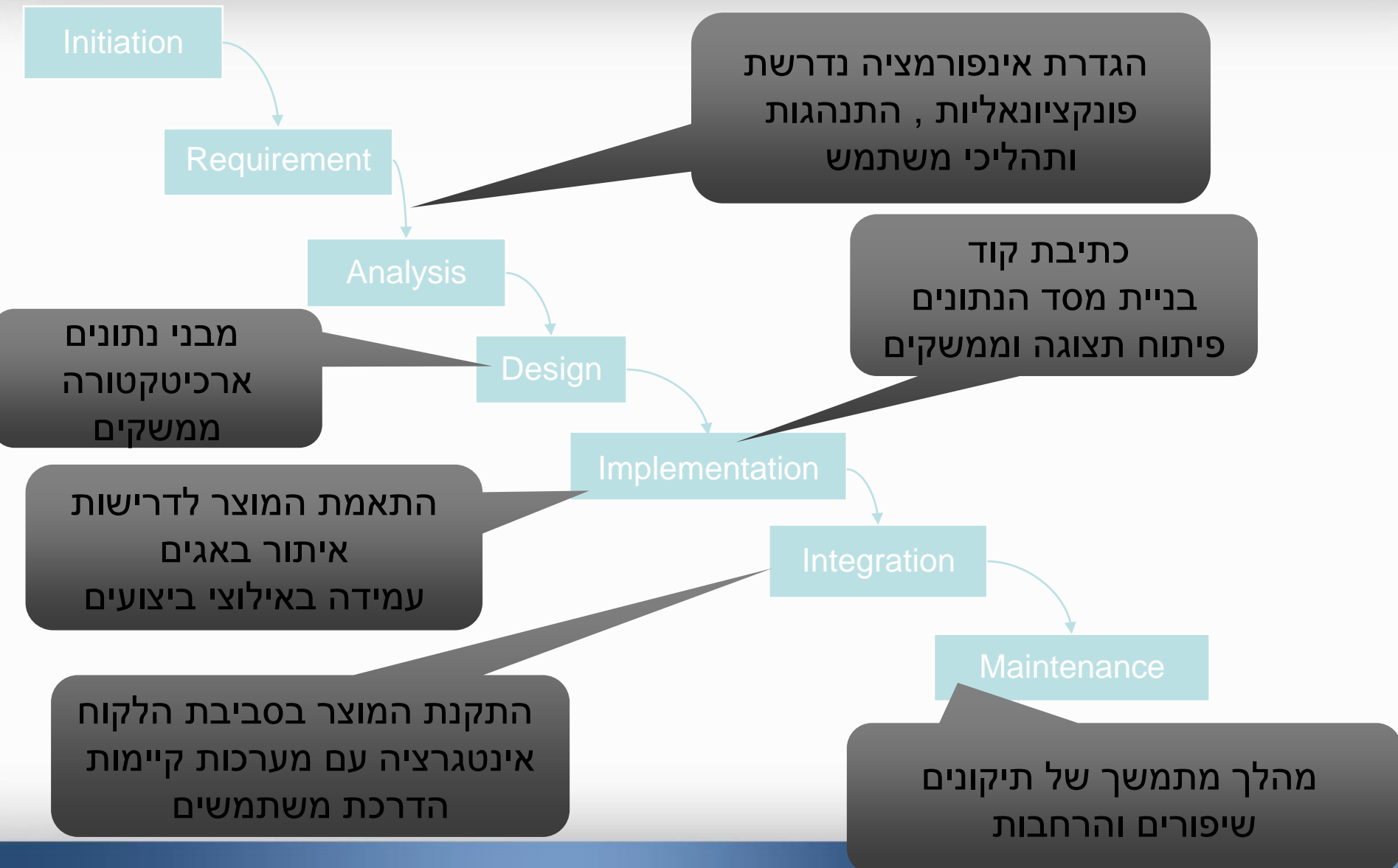
■ צוות תוכנה קטן או מתכנת בודד

- ניתוח ותכנון מינימלי
- אין הבחנה ברורה בין השלבים
- פתרון קטן העובד במהירות ובמינימום עלויות
- ללא תיעוד - התעלמות שיטתית משלבי התחזוקה

■ בעיות

- סינדרום ה -"משהו זמני"
- אם מדובר במערכת קריטית – מתכון לאסון
- עלויות שינויים ותחזוקה מזנקות עם הזמן
- אנטייתזה מוחלטת לרוח הקורס

Waterfall



- I believe in this concept, but the " –
implementation described above is risky

[illegible]

Figure 10. Summary

Waterfall

יתרונות

- מספק תהליך מובנה גם לחסרי ניסיון
- קל להבנה , פשוט לשימוש
- מספק יציבות לדרישות
- מקל על ניהול ושליטה בתהליך
- מכוון לקראת איכות ופחות אילוצי עלות או לוחות זמנים

חסרונות:

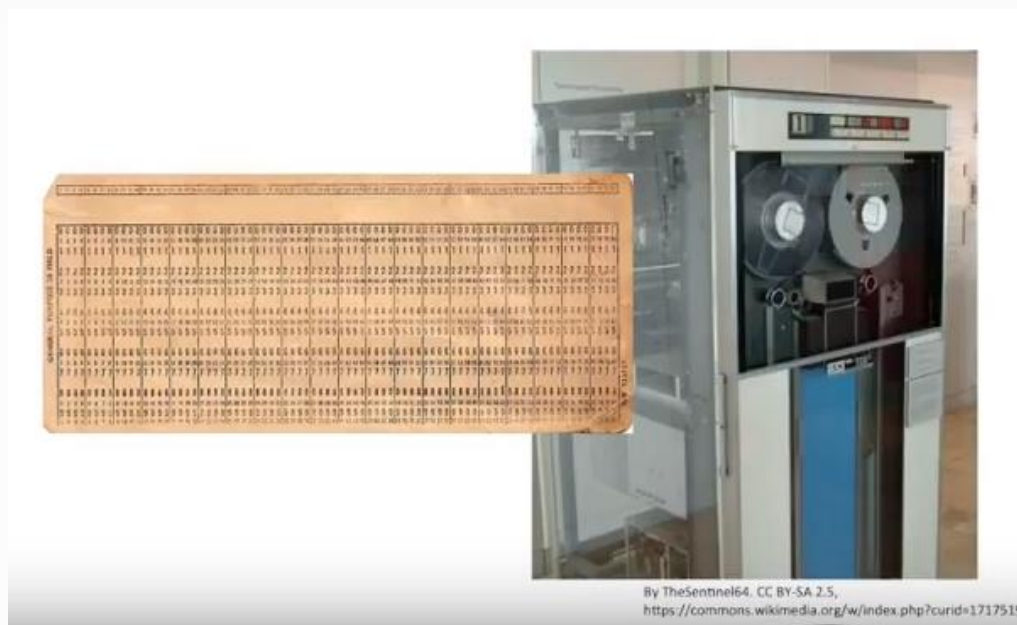
- יש להכיר את כל הדרישות ומראש
- תוצרי כל שלב אינם ניתנים לשינוי – הקפאת גרסאות
- העדפת גישת התהליך המובנה על פני "פתרון בעיות"
- שלב העברה לייצור קורה בשלב אחד גדול בסוף ולא במדורג
- מעט הזדמנויות או תחנות לקבלת פידבק מהלקוח

Waterfall

■ מתאים כאשר:

- הדרישות מאוד ברורות וידועות
- הגדרות המוצר מאוד יציבות
- הטכנולוגיה מובנת וידועה
- כאשר בונים גרסה חדשה למוצר קיים
- כאשר מייבאים גרסה קיימת לפלטפורמה חדשה

Waterfall



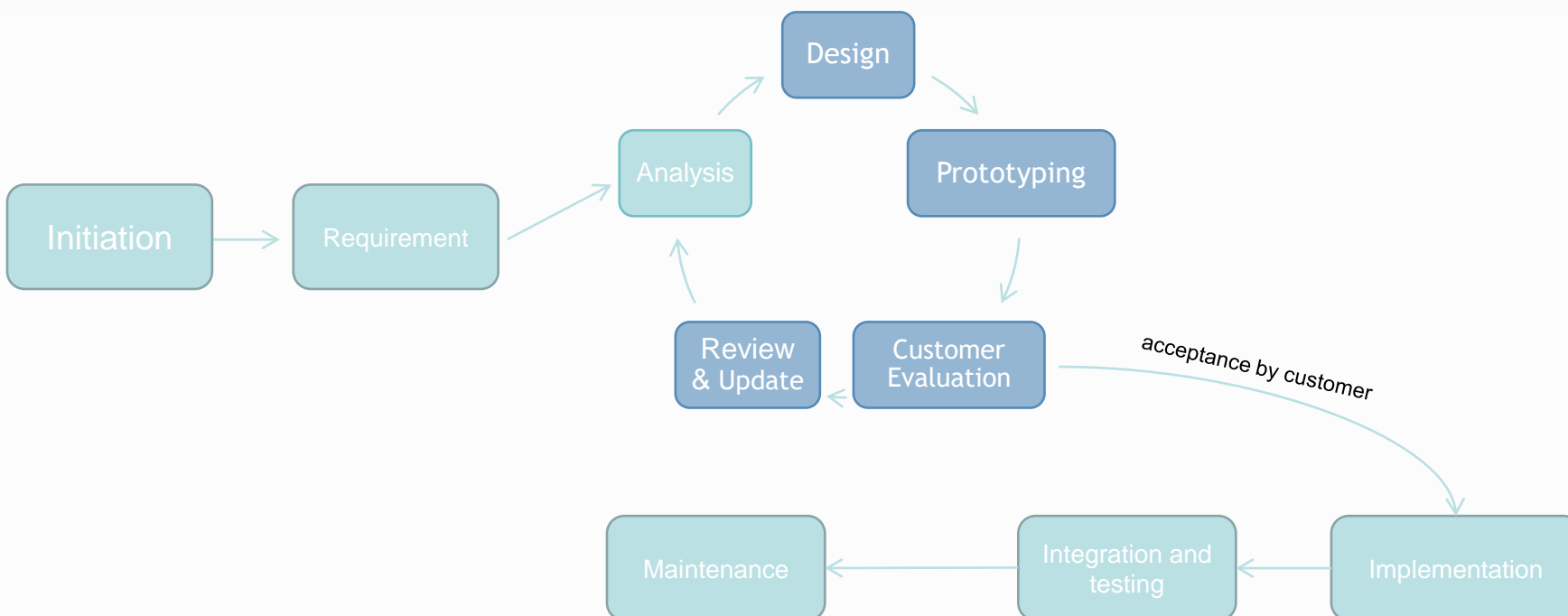
מודלים לינאריים נוספים

כניסיון מענה לבעייתיות של מודל מפל המים
פותחו כמה שיטות עבודה נוספות

- אב טיפוס

- V model

מודל אב טיפוס מהיר



מודל אב טיפוס מהיר

■ שלבים מאפיינים:

- המפתחים בונים אב טיפוס בשלב ניתוח הדרישות
- אב הטיפוס עובר ביקורת של משתמשי קצה
- משתמשי קצה מספקים משוב לתיקונים
- המפתחים משפרים את אב הטיפוס
- כאשר משתמשי הקצה מרוצים, ממלאים את שאר הפרמטרים כדי
להפוך אותו למוצר סופי

מודל אב טיפוס מהיר

- אב טיפוס- כשמו כן הוא – דגם של המוצר
 - יכולות מוגבלות
 - אמינות נמוכה
 - ביצועים לא יעילים
- משמש להדגמה בלבד, הוא לא המוצר הסופי!
- שימושי במיוחד כאשר-
 - דרישות המשתמש לא מלאות
 - נושאים טכניים לא לגמרי ברורים

מודל אב טיפוס מהיר

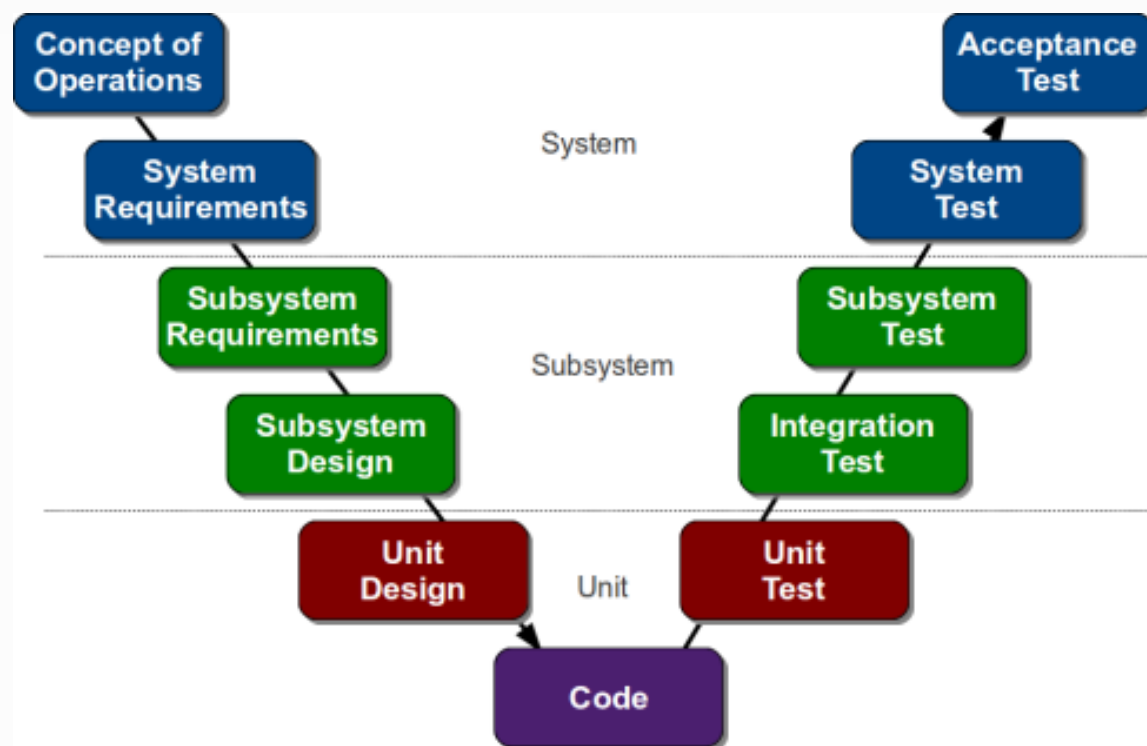
יתרונות

- הלקוחות יכולים לראות תרגום הדרישות למשהו מוחשי עוד בשלב האיסוף
- המפתחים לומדים ממשוב המשתמשים
- איתור דרישות ופונקציונאליות שלא היו צפויות
- מאפשר עיצוב גמיש
- קל לראות התקדמות

חסרונות:

- קיימת נטייה לאובדן מבנה התהליך
- ארכיטקטורה וראייה מערכתית אינם זוכים למקום מרכזי
- מתמקדים בריצוי מידי ודרישות תחזוקה נדחקות לשוליים
- קיימת סכנה לתהליך שלא נגמר (Scope creep)

V model



Definition and Specification

Implementation, Integration and Testing

V model

- וריאציה על מודל מפל המים.
- השלבים הראשונים של הפרויקט הם ניתוח ברמה כללית ובשלבים הבאים הניתוח נעשה מפורט עד רמת היחידה.
- לכל שלב מוגדרת אסופת בדיקות מקבילה
 - בחלק השני של המודל נמצאים שלבי הבדיקות בסדר הפוך- מתחילים בבדיקות ברמת היחידה, דרך בדיקות אינטגרציה ותתי המערכות ועד בדיקות המערכת ובדיקות קבלה סופיות

V model

■ יתרונות

- דגש על אימות ותיקוף המוצר בשלבים מוקדמים של התהליך
- כל תוצרי הביניים חייבים להיות בני בדיקה
- מנהלי הפרויקט יכולים לבחון התקדמות על בסיס אבני דרך
- קל להטמעה ושימוש

■ חסרונות

- מקשה על טיפול באירועים בו זמניים
- מקשה על התמודדות עם שינויים בדרישות

V model

מתאים כאשר:

- המערכת דורשת אמינות גבוהה ביותר (מערכת רפואית, עמידה בדרישות רגולטוריות קריטיות לתהליך)
- הדרישות ידועות ומוגדרות מראש
- הטכנולוגיה מובנת וידועה

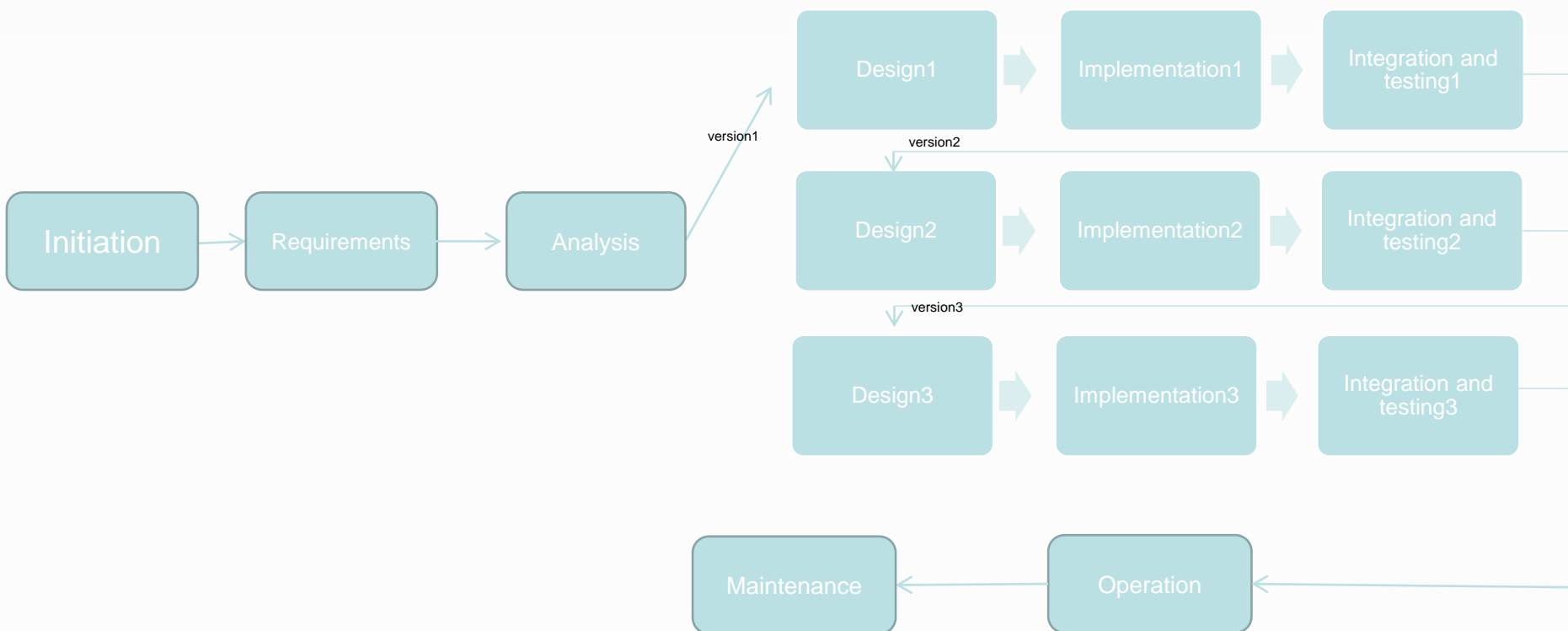
From linear to iterative

- קשה ומורכב לפתח תוכנה!
- נעשות טעויות במהלך פיתוח התוכנה
- הדרישות משתנות תוך כדי פיתוח
- כמו שהעולם כל הזמן משתנה כך גם מערכת התוכנה שאנחנו מפתחים



**מודלים קווים לא גמישים לשינויים ולכן נעבור
למודלים איטרטיביים**

Iterative Model



Iterative Model



<http://tryqa.com/what-is-iterative-model-advantages-disadvantages-and-when-to-use-it/>

Iterative Model

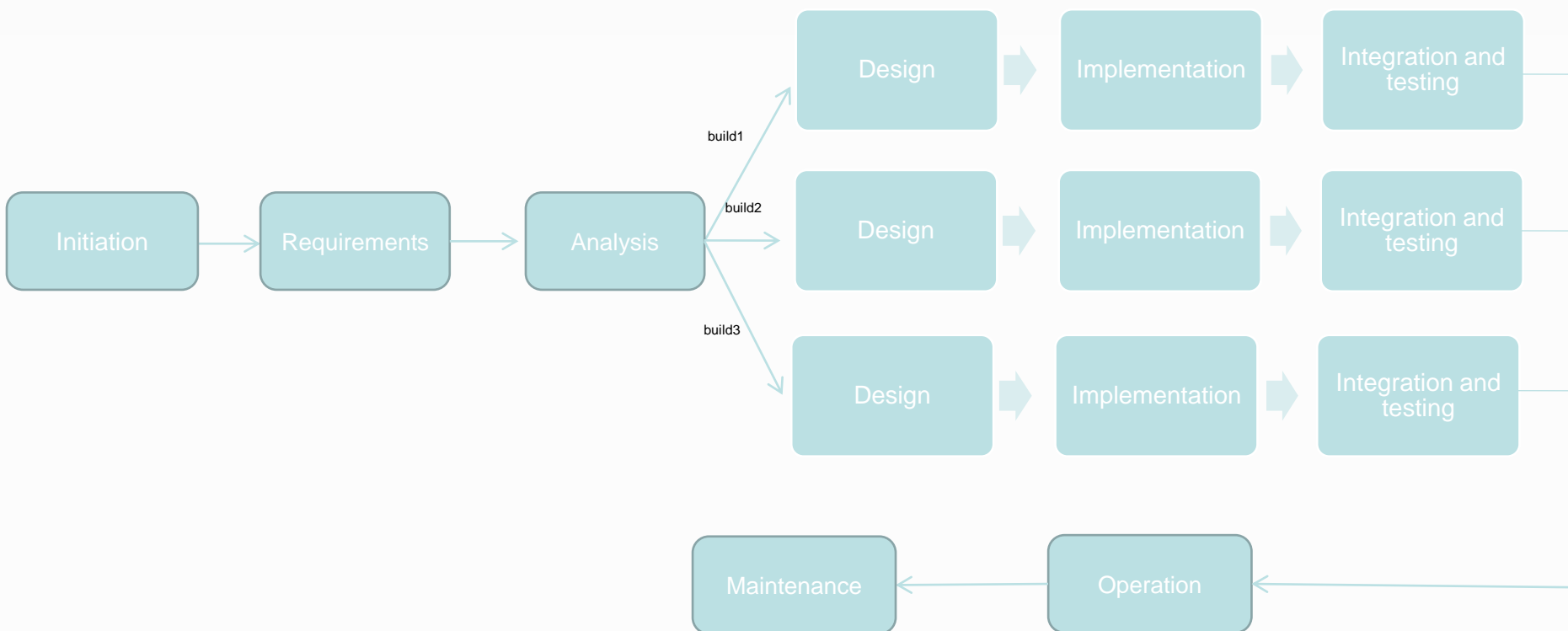
יתרונות:

- ניתן לראות תוצאות בשלב מוקדם של התהליך
- שינויים בדרישות עולים פחות
- יותר קל לבדיקה

חסרונות:

- דורש משאבים גבוהים
- אמנם עלויות השינויים נמוכים יותר, אבל עדיין לא ממש מתאים לשינויים בדרישות
- קשה לניהול -דורש ניהול צמוד וקפדני
- אין תמונה מלאה של המערכת הנדרשת לפני תחילת התהליך

Incremental model



Incremental model



<http://tryqa.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/>

Incremental model

יתרונות:

- יצירת תוכנה עובדת מהר ובשלב מוקדם של תהליך הפיתוח
- מודל גמיש יותר לשינויים, פחות יקר לשנות את התכולה והדרישות.
- קל יותר לבדוק איטרציות קטנות
- הלקוח יכול להגיב לכל מודול לחוד
- קל יותר לנהל סיכונים, כי חלקים מסוכנים מוגדרים ומטופלים באיטרציה שלהם.

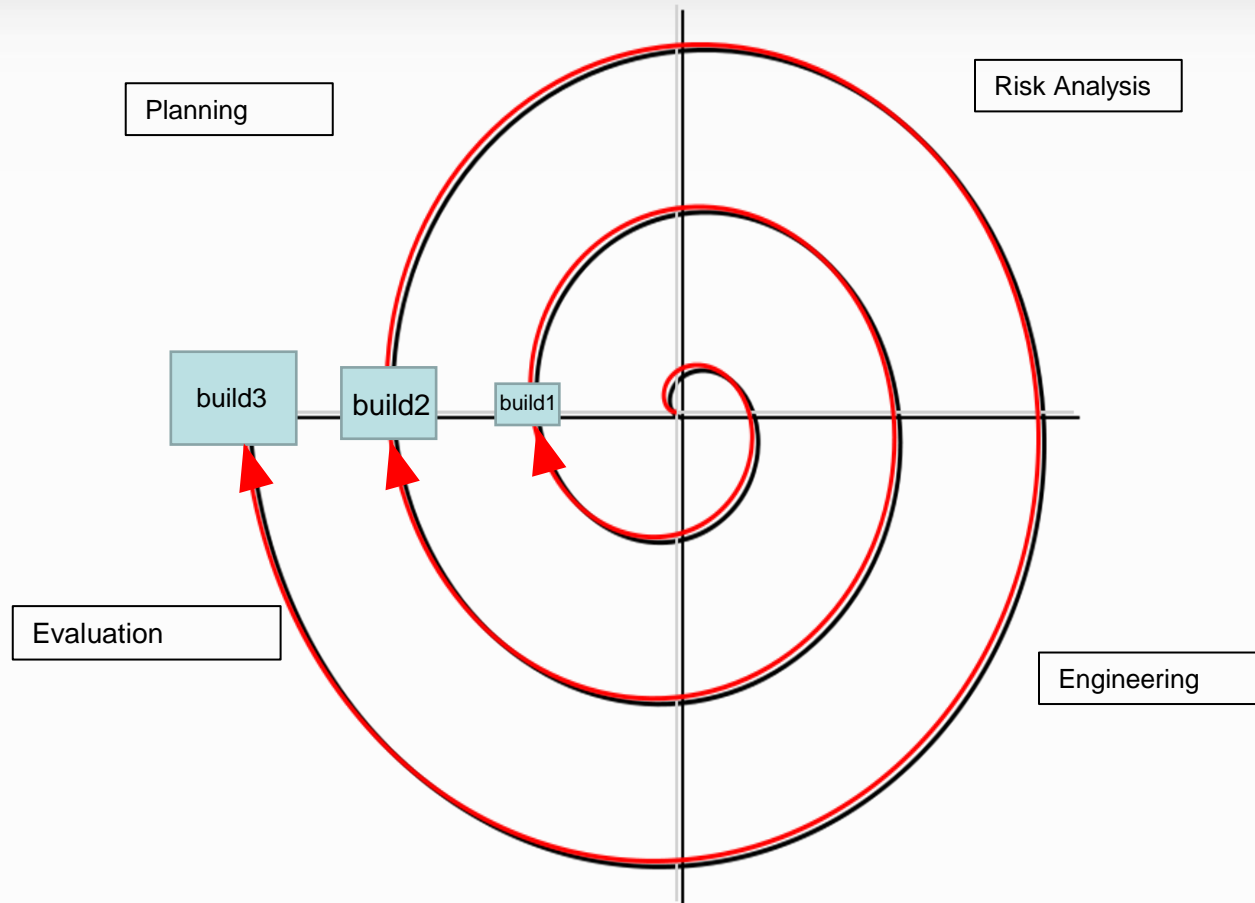
חסרונות:

- דרוש תיכנון ועיצוב קפדניים
- יש צורך להבהיר ולהגדיר את כל המערכת לפני שאפשר לחלק אותה למודולים
- העלות הכוללת כנראה תהיה גבוהה יותר ממודל מפל המים.

Spiral Model

- דומה למודל האינקרמנטלי אבל עם דגש לניתוח סיכונים.
- שלבי הפיתוח –
 - Planning –
 - Risk Analysis –
 - Engineering –
 - Evaluation –
- בתהליך פיתוח תוכנה עוברים באיטרציות על 4 השלבים, כאשר על איטרציה מבוססת על איטרציה הקודמת.

Spiral Model



Risk Analysis

ID	Risk	Probability	Loss	Risk Exposure	Risk Management Approach
1	Acquisition rates too high	5	9	45	Develop prototype to demonstrate feasibility
2	File format might not be efficient	5	3	15	Develop benchmarks to show speed of data manipulation
3	Uncertain user interface	2	5	10	Involve customer; develop prototype

Spiral Model

יתרונות ■

- גישות "בעיה ופתרונה" – העדפת התמודדות מול הדרישות ע"פ מבניות
- פיתוח פונקציות מרכזיות או פונקציות בעיתיות בראשונה
- אבחון סוגיות סיכון בשלבים ראשוניים במינימום עלות
- מסירה מהירה של גרסאות - כל גרסה כוללת מוצר עובד
- הלקוח יכול להגיב ולספק משוב תוך כדי הפיתוח
- הקטנת עלויות פיתוח ראשוניות
- התמודדות טובה עם סיכון הכרוך בשינויי דרישות תוך כדי הפיתוח

Spiral Model

■ חסרונות

- דורש יכולת ניתוח סיכונים
- העיצוב לא חייב להיות שלם מלכתחילה
- מודל מורכב, תקורות וסוגיות תזמון משאבים
- נדרשת הגדרת ממשקים טובה כמפתח לחיבור המרכיבים לגרסה הסופית
- עלות כוללת סופית לא פוחתת בהכרח

Spiral Model

■ מתאים כאשר:

- הגדרת סיכונים, מימון, לו"ז, מורכבות או הערכת תועלות נדרשת כבר בתחילת הדרך
- הדרישות ידועות ומוגדרות אך צפויות להתפתח ולהשתנות בהמשך
- בניית מוצר חדש
- פרויקטים בעלי סיכון גבוה
- הלקוחות אינם בטוחים מהם הדרישות
- הדרישות מורכבות
- יש מקום לפיתוח אב טיפוס

סיכום

- למדנו להגדיר דרישות
- ראינו שיטות שונות בתהליך הפיתוח
 - מפל המים – מבניות קשיחה , איכות מקסימלית
 - אב טיפוס מתפתח – הבהרת דרישות פרוגרסיבית
 - מודל V- תיקוף ועמידה בתקנים נוקשים
 - מודל ספירלי – ניתוח ובקרת סיכונים
 - מודל אינקרמנטלי – הגעה ראשונית מהירה לשווקים
- בשבוע הבא, נלמד שיטות מתקדמות בפיתוח תוכנה