



המכללה האקדמית להנדסה ירושלים

Henry Ford: “If I'd talked to my customers, they would have asked for faster horses.”

Berard : “Walking on water and developing software from a specification are easy if both are frozen”



F. P. Brooks, "The Mythical Man-Month":  
“The hardest single part of building a software system is deciding precisely what to build”

# הנדסת תוכנה

## 3. דרישות

# מקורות



- Pressman, Requirements 5-6
- Cockburn [Writing Effective Use Cases](#),
- More:
  - The Agile Samurai, Chap. 6
  - Pragmatic Programmer, p. 202-208
  - Survival Guide, Ch. 8: Requirements Development
  - Adzic, Bridging the Communication Gap, Specification by Example and Agile Acceptance Testing
  - Online tutorial:  
<http://www.cragssystem.co.uk/SFRWUC/index.htm>
  - Spolsky, "[Painless Functional Specifications](#)"

# קישורים

- Assaf Stove, [Why Should I INVEST in User Stories?](#), Blog 2012 (reading?)
- K. Henney, [Inside requirements](#) (req. classification), 2003
- Steve Yegge, [Business Requirements are Bullshit](#)
- [User Stories](#) by Mike Cohn, (INVEST [presentation](#))
- Dan North, [What's in a Story?](#)
- Gojko, [Agile product management using Effect Maps](#)
- Spolsky, [Painless Functional Specifications - Part 1: Why Bother?](#)

# איפה אנחנו בפרויקט (בקורס)?

- למה?  
בעיה (פלט: הצעת פרויקט\חזון\SOW)
- מי?  
צוות (Inception, אתחול\תכנון פרויקט)
- מה?  
דרישות (SRS)
- איך?  
תיכון (ארכיטקטורה) (SDS)
- מתי?  
תכנון וניהול – (ZFR)
- הלאה  
(איטרציות, Code)



# היום

- פרויקט – מה הלאה? השלמת קבוצות (זיהוי לקוחות)
  - משחק
  - דרישות
    - מפרט דרישות תוכנה
    - תרחישי שימוש
    - כלי מידול UML
    - סיפורי משתמש
    - מפרטים מורצים
  - סדנת דרישות
  - בתרגיל
- כתיבת מפרט דרישות SRS

# פרויקט – שלבים

- אבני דרך (עם תשלום), כל שבוע-שבועיים (שחרורים\גרסאות\..)

– Vision/SOW/SDP

– SRS

– SDS

– גרסת 0 (ZFR)?

– Backlog + ספרינטים

– גרסת בטא

– שחרור סופי

– סיכום

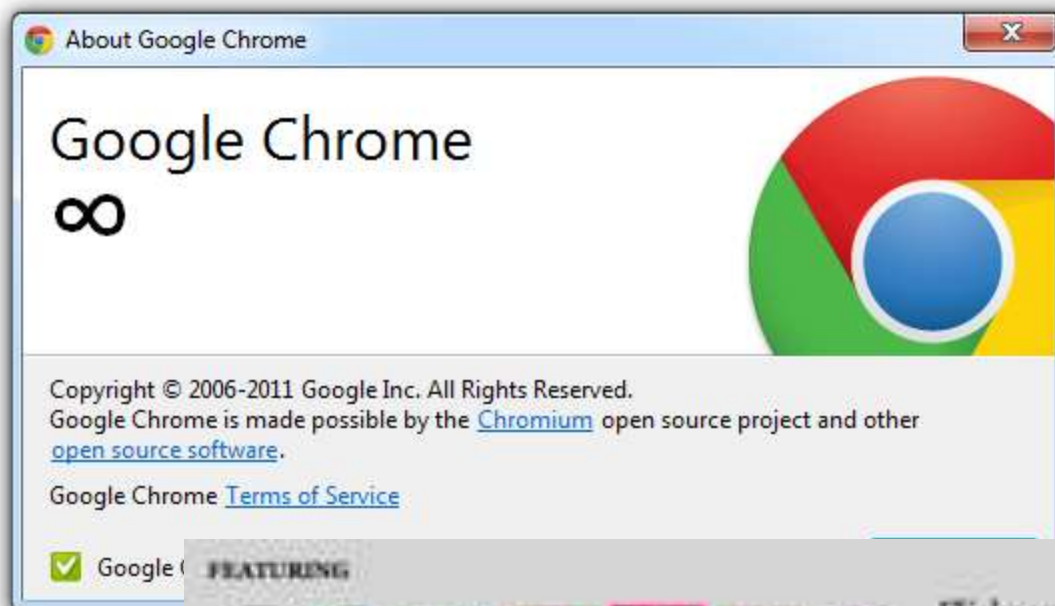
- הערכת עמיתים: הציון משקף הערכה חיצונית של הקבוצה לפי

אבני הדרך והתוצר הסופי, וגם את ההערכה הפנימית של

תרומת כל אחד לקבוצה ולפרויקט (חלקיות TBD)

Peer Assessment of Contributions of Group Members

# שחרורים?



**FEATURING**


boes

*Flickr was last deployed 15 hours ago, including 4 changes by 1 person.*

*In the last week there were 60 deploys of 471 changes by 19 people.*



# "אז מה עושים עכשיו\בינתיים?"

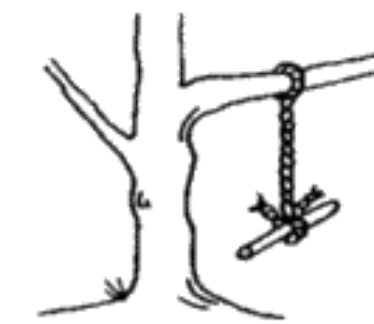


- מחקר טכנולוגי ועסקי
- נסיונות ([spikes](#)) להנמכת סיכונים (חלק מה-SRS, יש לתאם איתנו)
- הכנת תשתיות (חלק מהמשימה הבאה)
- דרישות: איסוף וניתוח

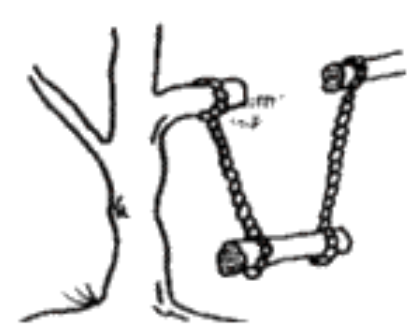


## Pragmatic Programmer Tip: **Don't Gather Requirements - Dig for Them**

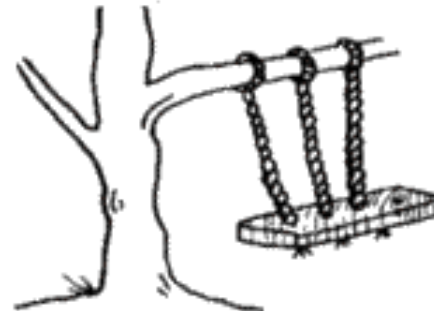
Requirements rarely lie on the surface. They're buried deep beneath layers of assumptions, misconceptions, and politics.



What the user asked for



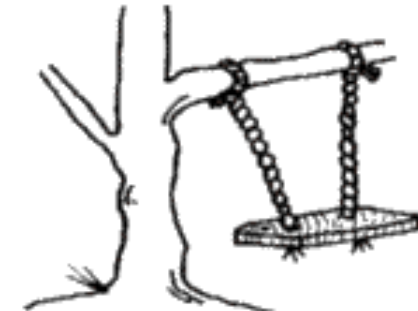
How the analyst saw it



How the system was designed



As the programmer wrote it



What the user really wanted



How it actually works

<http://www.youtube.com/watch?v=OfgfnZZdMII&feature=related>

# דרישות

# תהליך: פעילויות לדוגמא

- **דרישות**
    - איסוף, ניתוח (Analysis)
  - **תיכון (Design)**
  - **מימוש**
  - בדיקות\אימות\שילוב
  - תיעוד
  - הטמעה\תמיכה\אחזקה
- פעילויות תומכות
    - ניהול הפיתוח
    - הבטחת איכות
    - סקרים
    - ניהול תצורה
  - כלים

לכל פעילות בד"כ יש תוצרים

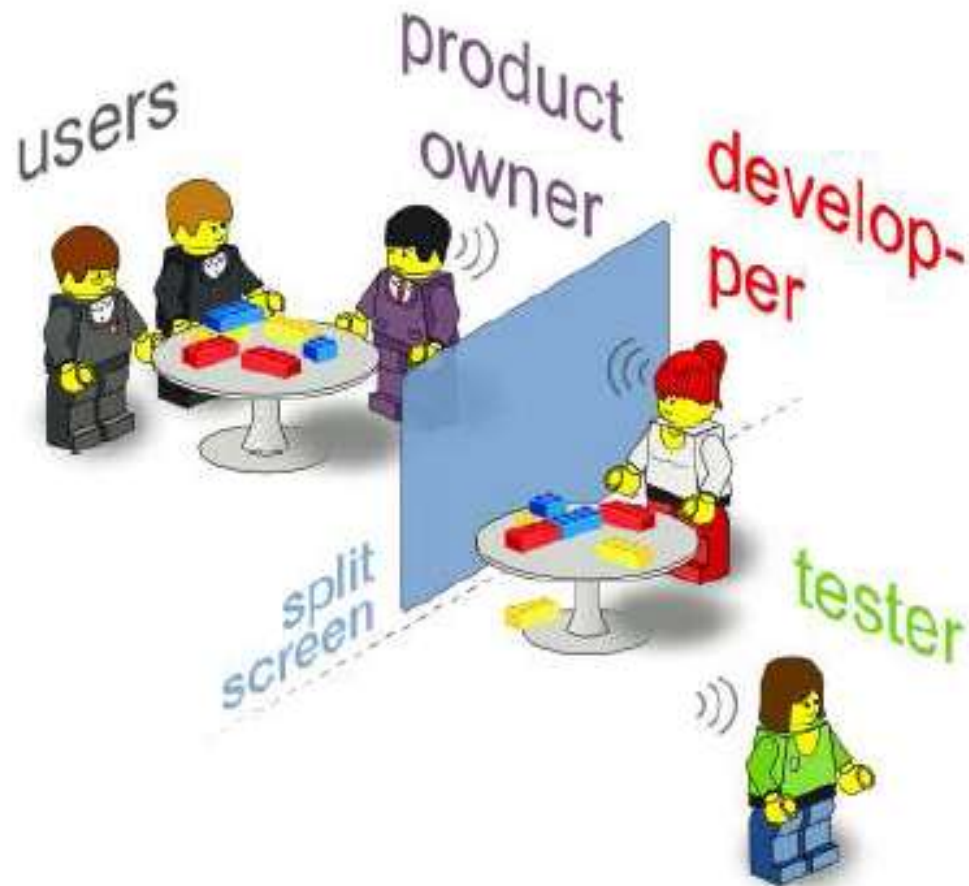
# ראשי פרקים

- כמה דברים מעניינים על דרישות
- מהן דרישות?
- איך אוספים דרישות?
- איך מגדירים דרישות?
- איך בודקים דרישות?

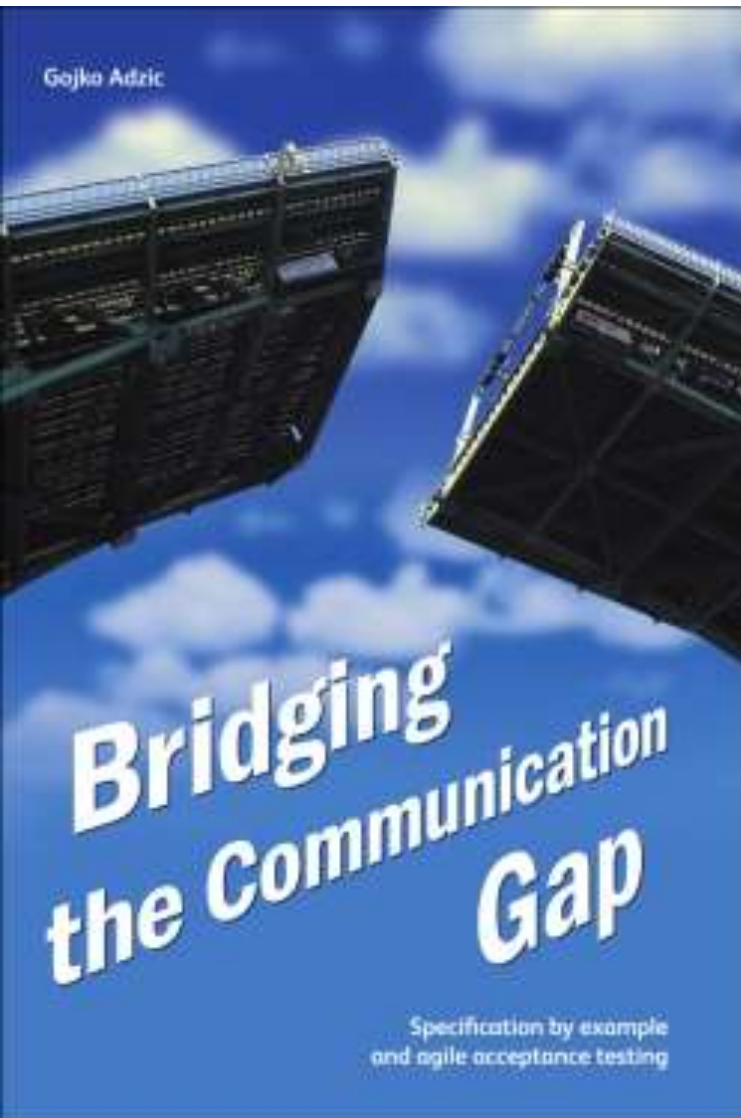
# משחק

- 99 Test Balloons

## עוד משחק (\*)



# פער התקשורת



- צדדים מעורבים
  - לקוחות \משתמשים
  - מפתחים\מהנדסים
- בד"כ הלקוחות לא מבינים בתוכנה והמפתחים לא מבינים את הבעיה (העסקית)
- מה קורה אם אחד מהצדדים יותר דומיננטי?
- כיצד מעבירים את הצרכים והידע אל מי שאמור לפתח?



והפתרון... (ניסיון 1)

מפרט דרישות תוכנה  
Software Requirement  
Specification

# דרישות תוכנה

- דרישות מגדירות ומפרטות, מה לבנות
  - את ה"מה" ולא את ה"למה" (SOW\חזון\הצעה) וה"איך"
  - התרכזות בבעיה ולא בפתרון
- למה צריך דרישות?
  - 
  - להבין בדיוק מה נדרש מהתוכנה
  - לתקשר עם כל המעורבים לצורך הבנה מדויקת
  - לבקר את התהליך, כדי לוודא שהמערכת מממשת את המפרט (כולל שינויים)





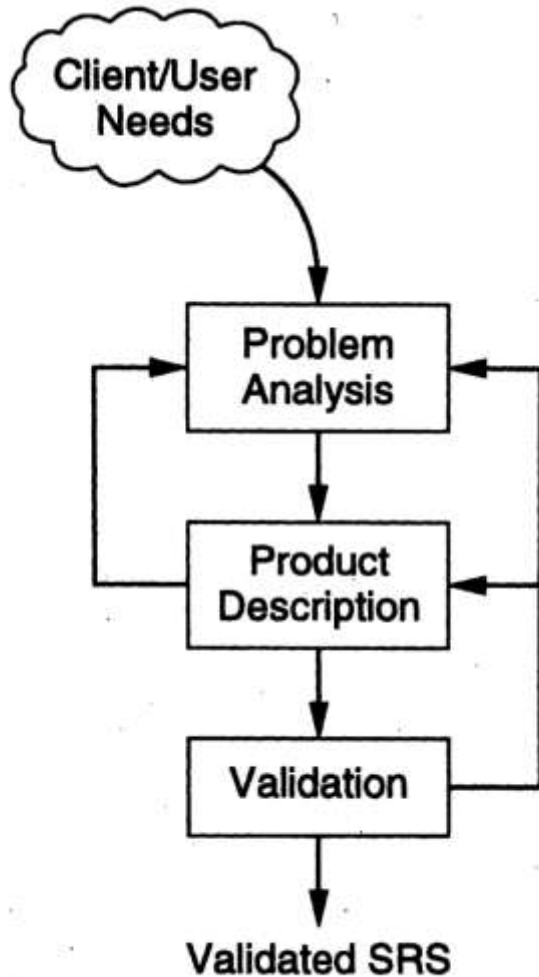
“The goal is NOT to write code. If we could ship products and make all this money without writing any code, we would. Your job is to *ship products EXACTLY on time*. It doesn’t matter whether you’re a developer, tester, program manager, product manager—whatever. Everybody’s job is the same.”

—Chris Peters, Excel 3.0

# מפרט ? Specification

- Lamport: a specification is a **contract** between user and implementer such that neither must talk to the other
- Brooks: clients do not and **can not know** their needs well enough to write such a contract
- Cockburn: A specification is notification that certain design decisions have been taken and the design space has been reduced. Each new specification should necessarily satisfy the previous; it may well happen that direct implementation is for many stages not practical.

# תהליך הדרישות



1. איסוף

2. ניתוח

3. פירוט

4. אימות

# שני סוגי דרישות עיקריים

- פונקציונליות \צרכים – השירות\ההתנהגות שהמערכת מספקת
- לא-פונקציונליות\אילוצי איכות – כל השאר
  - ביצועים (Atwood: [Performance is a Feature](#))
  - אמינות
  - פרטיות, אבטחה
  - תעוד, קלות שימוש
  - קלות הרחבה
  - סביבת הפעלה, ממשקים
  - עלות, זמנים
  - תקנים, חוקים
  - (כללי\איך \ilities ...\לא בהכרח למימוש – אז מי דואג להם?)

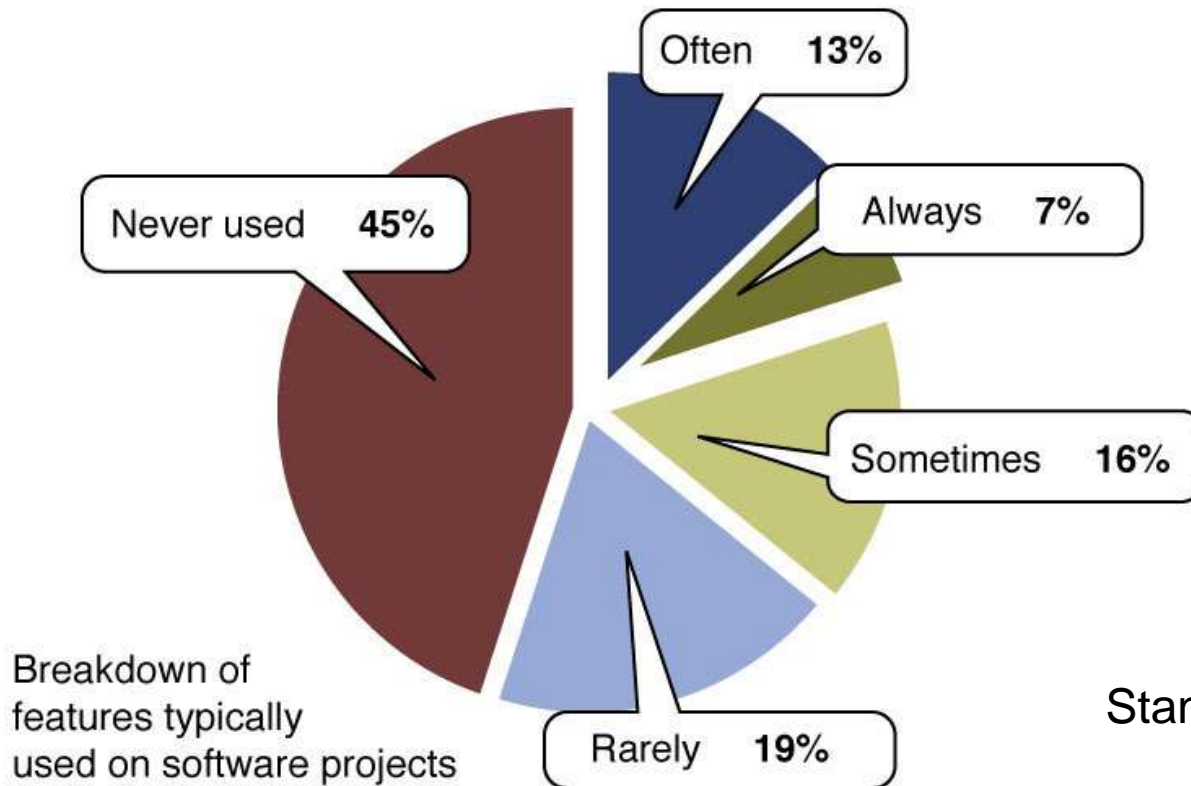
# אבטחה ואג'ייל

- בפרויקט חדש ברשת הזקוק לאבטחה משמעותית הועלו מספר טענות כנגד פיתוח בשיטות אג'ייל, מהי הטענה החלשה ביותר?

1. אבטחה היא דרישה לא-פונקציונלית ובלי מפרט דרישות ייתכן שהיא תתפספס
2. באג'ייל מתחשבים יותר מדי בלקוחות והם לא תמיד מבינים באבטחה
3. אבטחה מצריכה תכנון מראש ושיטתיות מה שחסר באג'ייל
4. הפיתוח הזריז באג'ייל גורם לחיתוך פינות ויכול לפגוע באבטחה

# הגדרה

- מילון: דבר נצרך עבור הקיום או ההתרחסות של דבר אחר
- האם?



Standish group 2002

# דרישה?

- Kent Beck: “Software development has been steered wrong by the term ‘requirement,’ defined in the dictionary as something that is mandatory or obligatory. The word carries a connotation of **absolutism** and permanence, inhibitors for embracing change. And the word ‘requirement’ is just plain wrong.
- “Out of the thousands of pages used to describe requirements, if you deliver the right 5, 10, or 20 percent, you will likely realize all of the **business benefit** envisioned for the whole system. So what were the other 80 percent? Not requirements—they weren’t mandatory or obligatory.”

• אג'ייל: דרישה <= מאפיין

# מהי דרישה טובה? (IEEE830)

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked
- Verifiable
- Modifiable
- Traceable



# איך אוספים דרישות?

ראשית:

- מחקר "כאוס" של קבוצת Standish על יותר מ-8000 פרויקטים מצא שהסיבה העיקרית לכשלון פרויקט, היא חוסר מעורבות של **המשתמש** (סיבה שניה: חוסר ניהול).
- "Easy access to **end users** is one of three critical success factors in rapid-development projects" (McConnell)

# יתרונות לעבודה עם לקוחות

- יחסים טובים מאפשרים פיתוח מהיר
- מאפשרים פיתוח לכיוון הנכון
- הם לא תמיד יודעים מה הם רוצים
- בעצם, הם כן יודעים, וזה משתנה עם הזמן
- ואפשר גם לעזור להם

## עוד כמה הגיגים

- The most difficult part of requirements gathering is not the act of recording what the users want; it is the exploratory, development activity of **helping users figure out** what they want. **McConnell**
- Work with a User to **Think Like a User** – it's the best way to get insight on how the system is easily used. **Pragmatic Programmer Tip**
- I like to listen. I have learned a great deal from **listening carefully**. Most people never listen. **Ernest Hemingway**

I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.



E-mail: SCOTTADAMS@AOL.COM

FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?



I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.



I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?



© 2006 Scott Adams, Inc. / Dist. by UFS, Inc.

I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.



TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!



CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?



www.dilbert.com

# איך אפשר להגדיר דרישות?

- אז... אנחנו עובדים עם הלקוח להבין את הצרכים שלו, איך לוכדים את הדרישות האלו? מה האפשרויות?
- אב-טיפוס
- משחקי תפקידים, סיעור מוחות
- ראיונות, שאלונים, ביקור בית
- **SRS – מפרט דרישות תוכנה**
  - תרחישי שימוש Uses Cases
  - ביצועים, אילוצים, ...
  - ממשקים (משתמש וחיצוניים)
  - טבלת דרישות
- **אג'ייל: סיפורי משתמש, סקיצות UI, מפרטים מורצים**
- שיטות פורמליות (Petri Nets, FSM, Z)

“Documents are worthless, but documenting is everything”. (Gause & Weinberg following [Eisenhower Quote](#) on planning)

# Use Case – תרחיש שימוש

- מתאר דרך מסוימת להשתמש במערכת
- מייצג דו-שיח בין משתמש והמערכת מנקודת הראות של המשתמש -קופסה שחורה
- כלי ללכידת דרישות פונקציונליות
- פותחו ב- UML (ג'יקובסון, קוברן '90) פורמליות "רכה"
- “A specific way of using the system by using some part of functionality”, Jacobson

# הגדרות

- **שחקן Actor**: מישהו שבא במגע עם המערכת
- **בעל ענין Stakeholder**: מישהו שיש לו ענין במערכת המפותחת (SuD)
- **Use Case\תרחיש שימוש**: חוזה על דרך פעולתה של המערכת
- **שחקן ראשי**: זה שמניע את התרחיש (UC)
- **שם\מטרה**: התוצאה הרצויה לשחקן הראשי ולבעלי הענין.
- **הקף ורמה**: הפרוט המופיע בתרחיש והיעד: ארגוני-מערכתי-תת-מערכתי

# סוגי חלקי UC

- טבלת שחקנים
- דיאגרמת תרחישים, נתמכת בכלים שונים (UML):
  - IBM Rational Rose, Enterprise Architect, Rhapsody
  - MS Visio, MS Visual Studio 2010
  - Free: ArgoUML / UMLet / StarUML
  - Online: [yUML](#), [WebSequenceDiagrams](#), [gliffy](#)
- מבט על חלקי כלל תרחישי המערכת
- תרחיש לא פורמלי
- תרחיש פורמלי (UML)
- ובהמשך: סיפורי משתמש

בואו נבחן את הסוגים השונים:



# 1. תרשימי (סיכום) תרחישים

- סיכום של התרחישים יכול להופיע בתרשים
  - שחקנים: בצורת אייקון **בן-אדם**, עם שמם (שם עצם)
    - עדיין יתכן שמדובר במערכות מחשב חיצוניות
  - תרחישים: בצורת **אליפסה**, עם שמם (פעולה – משהו שהשחקן רוצה להשיג)
  - **קווי קישור**, מקשרים את השחקנים עם התרחישים שהם משתמשים בהם (חץ עבור השחקן הראשי)
  - **גבולות המערכת כמלבן** מסביב לתרחישים
  - **הקשרים** בין הגורמים השונים
- תרחישים יכולים גם להיות מחוברים לתרחישים שהם משתמשים או תלויים בהם



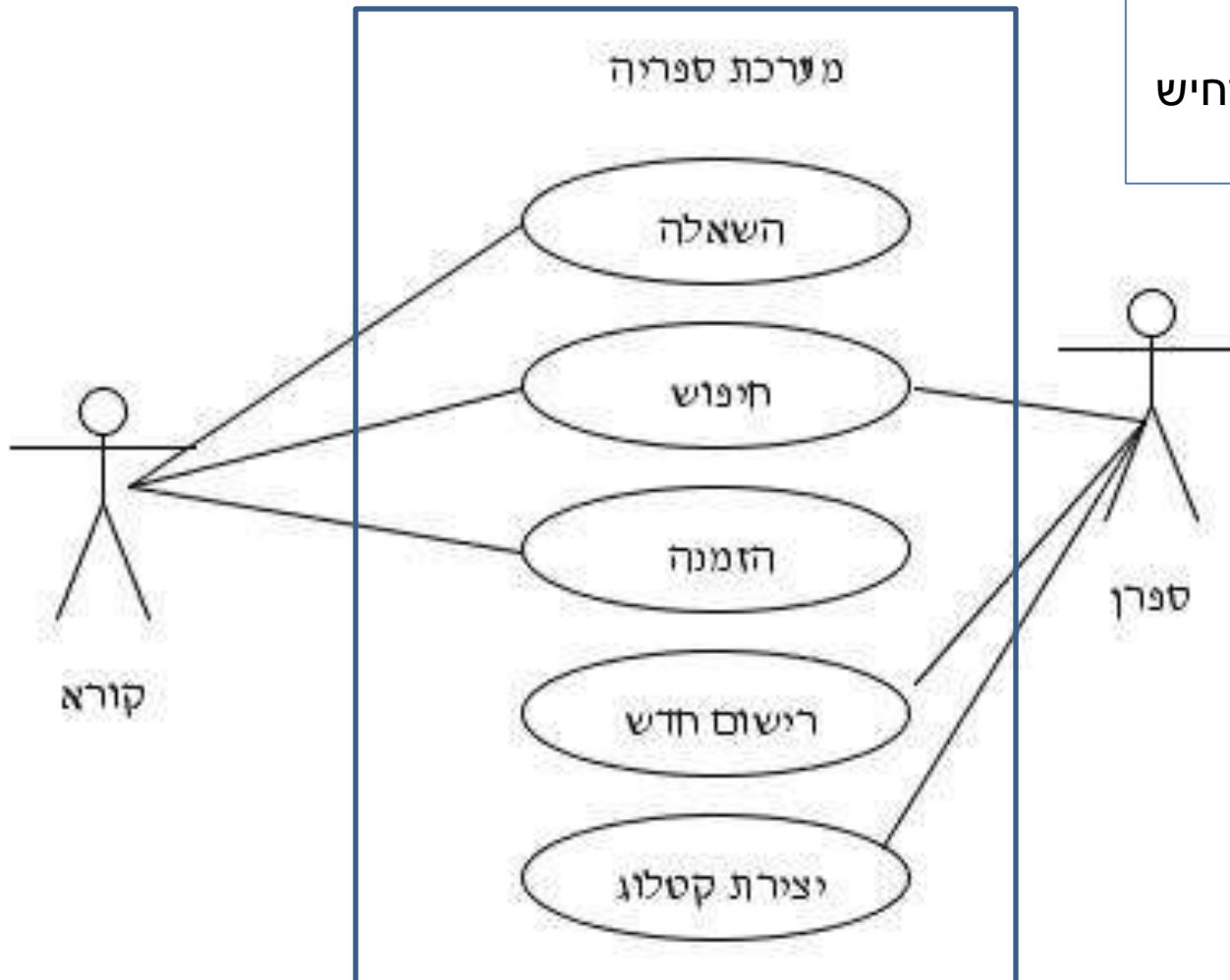
# טבלת שחקנים ובעלי עניין

- בד"כ מצורפת גם טבלה, של שחקנים ראשיים ומטרותיהם מהמערכת ז"א התרחישים שהם מניעים (מומלץ גם להבין את מטרות כל בעלי העניין)

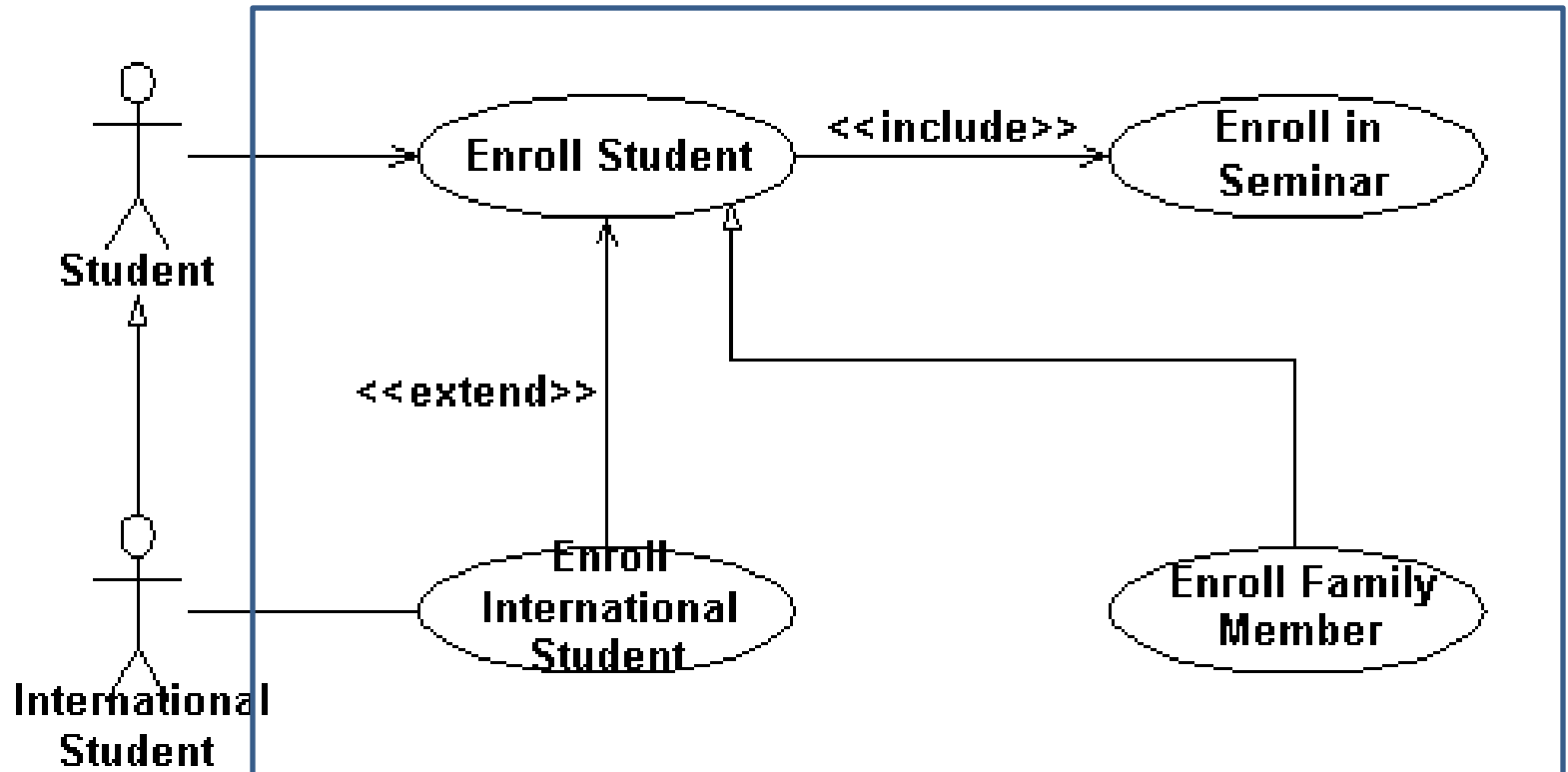
שחקן	מטרות (ותרחישים)
קורא	חיפוש ספר
	השאלת ספר
	החזרת ספר
ספרן	חיפוש ספר
	בדיקת המצאות ספר
	בקשת ספר מספריה אחרת
המכללה	שירות לסטודנט

# תרשים תרחיש

איפה צריך לשים חיצים?  
מה צריך להוסיף עבור תרחיש  
השאלה בין ספרייתית?



# תרשים תרחישים – עם קשרים



# תרשים UC - יתרונות

- זיהוי שחקנים
- מבט כללי
- נקודת התחלה

## 2. תרחיש לא-פורמלי

- נכתב כפסקה המתארת תרחיש\אינטראקציה מלא

– עם נתונים ספציפיים

- דוגמא:

קורא מאבד ספר

**הקורא** מדווח לספרן שהוא איבד ספר. **הספרן** מדפיס את רשומת הספר ומבקש מהקורא לדבר עם **מנהלת הספרייה**, שתקבע את גובה התשלום. **המערכת** תעודכן בנתוני הספר שאבד וכן כרטיס הקורא. **מנהלת הספרייה** עשויה להורות על אי-תחליף.

### 3. תרחיש פורמלי – דוגמא הזמנת ספר

שם	הזמנת ספר
שחקן ראשי	קורא
מטרה	קורא מעוניין לשריין ספר מתוך הקטלוג המקוון
הקף	מערכת הספריה
רמה	משתמש
בעלי עניין ואינטרסים	קורא – לשריין ספר בעל הספריה – שרות מורחב לרווחת הלקוחות
טריגר	הקורא נכנס למערכת
תנאי-קדם	הקורא עבר את מסך ההזדהות (login) ונחת בעמוד הבית
תנאי סיום מוצלח	הספר שמור עבור הקורא
תנאי כישלון	הספר אינו שמור

<p>1. הקורא לוחץ בתפריט על הזמנת ספר</p> <p>2. המערכת מציגה קטלוג עם מסך חיפוש</p> <p>3. הקורא מזין את שם הספר</p> <p>4. המערכת מציגה התאמות עם מיקומם</p> <p>5. הקורא בוחר התאמה ובקשה לשמירה</p> <p>6. המערכת מאשר את ההזמנה ומציגה את הקטלוג בחזרה</p>	<p>תרחיש הצלחה עיקרי</p>
<p>2א. פג תוקף ה-login</p> <p>2א.1. המערכת מחזירה את הקורא למסך הכניסה</p> <p>2א.2. הקורא מתייash או מנסה שוב</p> <p>4א המערכת אינה מוצאת את הספר</p> <p>5א.1. ...</p>	<p>הרחבות (שגיאות)</p>
<p>3. הקורא מזין מחבר או נושא</p>	<p>תרחישים חלופיים</p>



# צעדים ליצירת תרחיש ביצוע

1. זיהוי שחקנים ומטרותיהם

א- אלו אנשים, מכונות ומערכות נוספות יהיו בקשר עם המערכת שלנו (שחקנים)

ב- מה כל שחקן צריך שהמערכת שלנו תבצע

ג- כדאי גם לפרט את המטרות של בעלי עניין אחרים

2. יצירת דיאגרמת תרחישים

3. פירוט לתרחישים פורמליים \ לא-פורמליים

# יתרונות לתרחישי שימוש

- מובן ללקוח אך עדיין פורמלי
- יצירת הבנה בין הלקוח והמפתחים בקשר לדרישות (תרחישי הצלחה)
- חושף את המפתחים לנושאים בעייתיים (תרחישי הרחבה, חריגות)
- מאפשרים לתעדף מאפיינים ולתכנן בהתאם
- משמשים כקלט להמשך הפרויקט (הערכה, בדיקות, QA, ניהול)

# תרגיל



- בואו נזהה שחקנים ומטרות  
עבור הפרויקטים שלכם
- תרשים תרחישים ראשוני

### 3. כתיבת תרחיש הצלחה

- תרחיש ההצלחה העיקרי, הוא המסלול המועדף כשהכל הולך חלק
  - הקל ביותר לקריאה והבנה
  - כל השאר הם הסתעפויות וסיבוכים
- מתאר את מהלך העבודה של שחקן מהטריגר ועד לביצוע משימתו
  - צעדים ברורים וממוספרים

## 4. הרחבות כשלון

- בד"כ כמעט כל שלב יכול להיכשל
- מציינים זאת לאחר תרחיש ההצלחה
- אך מקשרים למספר הצעד הרלוונטי

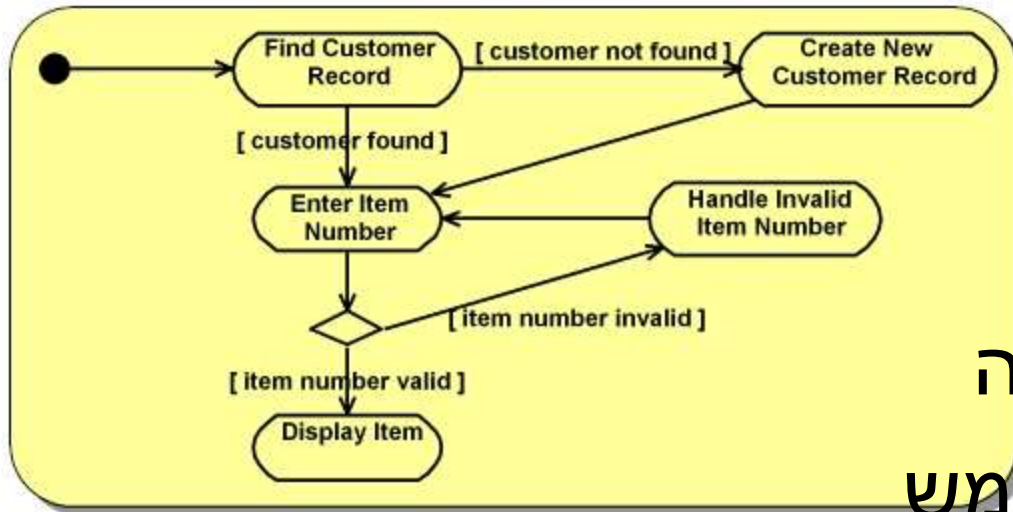
## 5. חלופות אחרות

- להרבה צעדים יכולה להיות התנהגות חלופית
- מקשרים גם כן למספר הצעד
- למשל:
  - 5'. חלופה ראשונה לצעד 5
  - 5". חלופה שניה לצעד 5

# מאפייני תרחיש ביצוע טוב

- מתחיל בפניה של שחקן למערכת
- נגמר במענה לכל צרכי הבקשה
- מגדיר את יחסי הגומלין (בין השחקן והמערכת)  
שקושרים לבקשה
- נכתב מנקודת ראות של השחקן ולא המערכת
- מתרכז ביחסים ולא בפעולות פנימיות של המערכת
- אין תאור מפורט של GUI
- תרחיש הצלחה עיקרי בעל 3-9 שלבים
- נוח לקריאה (נכנס בעמוד)

# מה עוד מכיל מפרט דרישות



- דיאגרמות פעילות
- מכוונות מצבים
- ממשקי חומרה ותוכנה
- תרשימי ממשק משתמש

Enter Surname	Customer Record
<input type="text" value="Brown"/>	Mr Andrew Brown 23 Northfield Crescent Wigan WN23 4HG
<b>Select Record</b>	Credit Available: £2345.67p YTD Purchases: £432.19p
<div>Brown, Andrew - Wigan</div> <div>Brown, George - Wigan</div> <div>Brown, Jean - Harrow</div> <div>Brown, Michael - London</div>	
<input type="button" value="Accept"/>	<input type="button" value="Create New Customer"/>

Customer Selection Screen



# היה היו דרישות...

- Alberto Savoia, **GTAC 2011: Opening Keynote Address - Test is Dead**  
<http://www.youtube.com/watch?v=X1jWe5rOu3g&t=5m48s>

• והיום...

# תשובה 2 אג'ייל: סיפורי משתמשים

- הגדרת דרישות ברמה כללית בלבד
- כוללים מספיק מידע שיאפשר הערכה למימוש
- בד"כ קצרים מתרחישי שימוש
  - יכולים להיות כותרת של תרחיש
- תזכורת לשיחה עם הלקוח, דוגמאות:
  - הקורא יכול להשאיל ספר באופן מקוון
  - המערכת שולחת תזכורת במייל כאשר תאריך ההשאלה פג
- [Cockburn](#): “promissory notes for future conversation”

# Ron Jeffries: CCC

## Card •

- סיפורים נכתבים על כרטיסיות
- מוסיפים עליהם הערכה, עדיפות ועוד

## Conversation •

- הפירוט מגיע בשיחה עם הלקוח

## Confirmation •

- בדיקות קבלה מאשרות שהסיפור מומש נכון

# תבנית לסיפור

- בתור ...
- עבור מי הסיפור
- אני מעוניין ....
- מה הוא\הם רוצה לעשות
- למה הוא רוצה לעשות זאת
- כך ש ...

# לדוגמא - רכישת דיסק

- בתור חובבת מוסיקה
- אני מעוניינת לראות את הכותרים  
האחרונים
- כך שאוכל להזמין ולהנות מהדיסק של...

# INVEST in User Stories

- [Bill Wake]:  
Independent, Negotiable, Valuable,  
Estimatable, Small, and Testable.
  - עצמאי, פתוח למשא ומתן, בעל ערך ללקוח,  
ניתן להערכה, קטן, ניתן לבדיקה
  - (עדיף מקצה לקצה)
  - (ר"ת אחרים S.M.A.R.T)

# אנטי-דוגמאות

- "אוטמציה של שרת ה-build"  
– ללא ערך ללקוח
- "דיפלוימנט לשרת סטייג'ינג מחוץ לפיירוול"  
– פרטי מימוש ללא התוצאה, טרמינולוגיה שהלקוח אינו מכיר -> "פרסום דמו שהלקוח יכול להשתמש בו"
- "האתר עולה מהר"  
– חסר קריטריון ברור להערכה -> "החיפוש מסתיים תוך שניה"

# אילו מהבאים הוא הכי פחות INVEST

1. המשתמש יכול לחפש סרט לפי כותרת
2. לאתר הסרטים צריכה להיות תגובתיות טובה
3. כשמוסיפים סרט למאגר ב-99% מהזמן הדף צריך להחזיר תשובה בפחות מ-3 שניות
4. כלקוח, אני רוצה לראות את רשימת הנמכרים ביותר ממוינת לפי מחיר, כך שאוכל לקנות קודם את הזולים



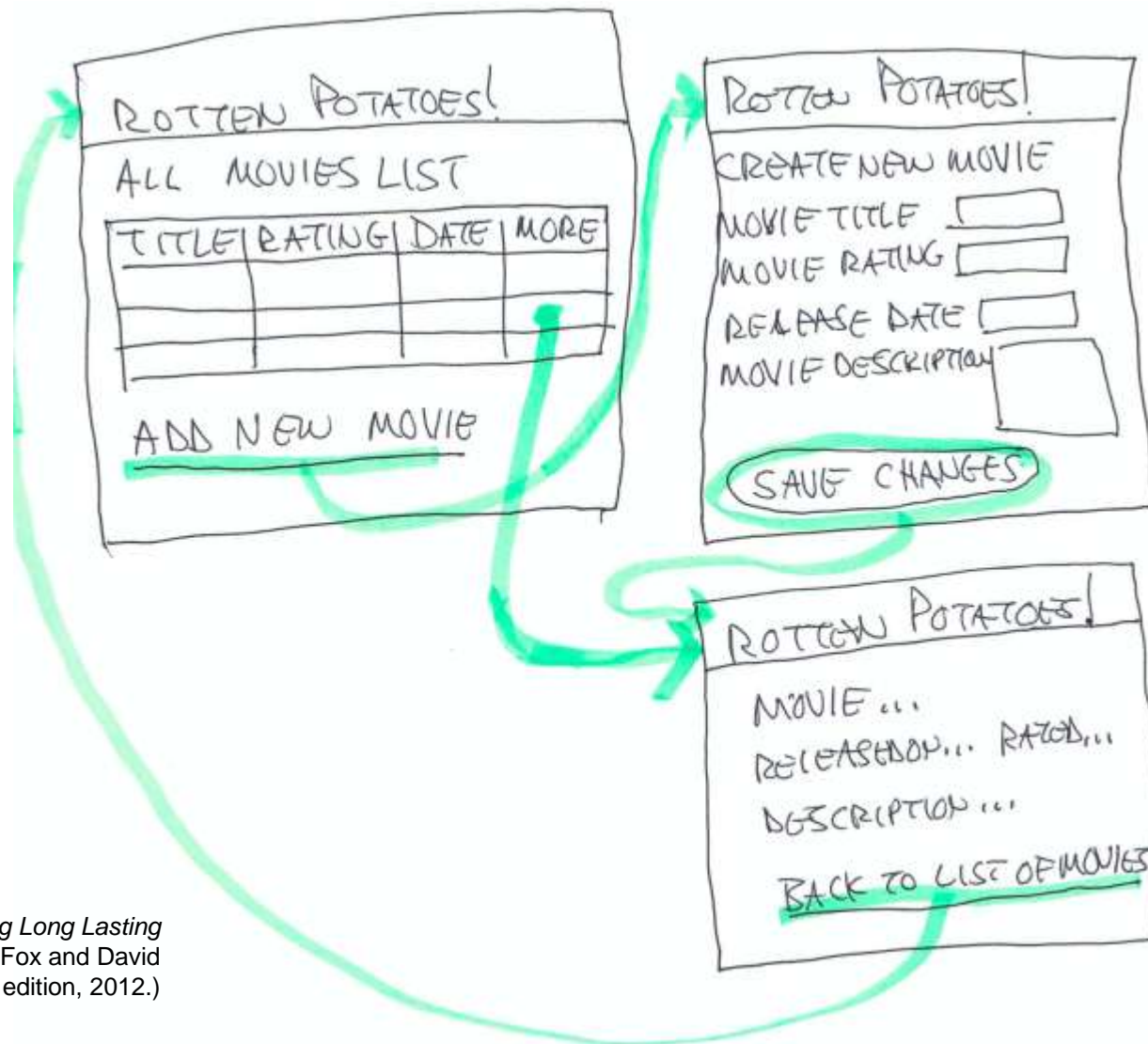
# תרגיל

- נסחו 3-4 סיפורי משתמש לפרויקט שלכם
  - כותרות בלבד
  - בדקו שהם INVEST

# סקיצות ו-Storyboard

- הדגמת שינויי ממשק בעקבות פעולות המשתמש
- תחום ממשק אדם-מכונה (HCI)
  - בדומה לתכנון סצנות בסרט
- עוד כלי להעצמת צד המשתמש

# Storyboard - לוגי



(Figure 4.4, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

# תרגיל

- שרטטו מסך עיקרי ראשוני למוצר שלכם
- תכננו מעברים למסכים אחרים

# האם הדרישות נכונות?

- מה יכול להשתבש?

- 

- חוסרים

- סתירות

- טעויות עובדתיות

- דו-משמעות

- מה עושים?

# כיצד מאמתים דרישות?

- סקר
- ניתוח
- אב-טיפוס
- יצירת מפרט בדיקות
- כלי טקסט ( NASA Automated Requirement Measurement )
- הרצת הדרישות (בשיטות פורמליות)
- לאחרונה: מפרטים מורצים (BDD, AAT) ..

# ~Demo: Cucumber

## 1: Describe behaviour in plain text

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

Scenario: Add two numbers
  Given I have entered 50 into the calculator
  And I have entered 70 into the calculator
  When I press add
  Then the result should be 120 on the screen
```

## 2: Write a step definition in Ruby

```
Given /I have entered (.*) into the calculator/ do |n|
  calculator = Calculator.new
  calculator.push(n.to_i)
end
```

## 3: Run and watch it fail

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
Scenario: Add two numbers # features/addition.feature
  Given I have entered 50 into the calculator # features/step_definitions/calculator_steps.rb:2:in 'Given /
  uninitialized constant Calculator (NameError)
  ./features/step_definitions/calculator_steps.rb:2:in 'Given /
  features/addition.feature:7:in 'Given I have entered 50 into
  And I have entered 70 into the calculator # features/step_definitions/calculator_steps.rb:3:in 'And I have entered 70 into the calculator
  When I press add # features/step_definitions/calculator_steps.rb:4:in 'When I press add
  Then the result should be 120 on the screen # features/step_definitions/calculator_steps.rb:5:in 'Then the result should be 120 on the screen'
```

## 4. Write code to make the step pass

```
class Calculator
  def push(n)
    @args ||= []
    @args << n
  end
end
```

## 5. Run again and see the step pass

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
Scenario: Add two numbers # features/addition.feature
  Given I have entered 50 into the calculator # features/step_definitions/calculator_steps.rb:2:in 'Given /
  And I have entered 70 into the calculator # features/step_definitions/calculator_steps.rb:3:in 'And I have entered 70 into the calculator
  When I press add # features/step_definitions/calculator_steps.rb:4:in 'When I press add
  Then the result should be 120 on the screen # features/step_definitions/calculator_steps.rb:5:in 'Then the result should be 120 on the screen'
```

## 6. Repeat 2-5 until green like a cucumber

```
$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
Scenario: Add two numbers # features/addition.feature
  Given I have entered 50 into the calculator # features/step_definitions/calculator_steps.rb:2:in 'Given /
  And I have entered 70 into the calculator # features/step_definitions/calculator_steps.rb:3:in 'And I have entered 70 into the calculator
  When I press add # features/step_definitions/calculator_steps.rb:4:in 'When I press add
  Then the result should be 120 on the screen # features/step_definitions/calculator_steps.rb:5:in 'Then the result should be 120 on the screen'
```

Feature: Addition

In order to avoid silly mistakes

As a math idiot

I want to be told the sum of two numbers

Scenario: Add two numbers

Given I have entered 50 into the calculator

And I have entered 70 into the calculator

When I press add

Then the result should be 120 on the screen



```

$ cucumber features/addition.feature
Feature: Addition # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
Scenario: Add two numbers # features/addition.feature
  Given I have entered 50 into the calculator # features/step_definitions/calculator.rb:2:in `Given I have entered 50 into the calculator'
  And I have entered 70 into the calculator # features/step_definitions/calculator.rb:2:in `Given I have entered 70 into the calculator'
  When I press add # features/step_definitions/calculator.rb:3:in `When I press add'
  Then the result should be 120 on the screen # features/step_definitions/calculator.rb:4:in `Then the result should be 120 on the screen'

```

# מתרחישים לטבלת דרישות

- תרחישים  $\leq$  (בעיקר) כלי ניתוח
- טבלת דרישות \ Backlog \ URD  $\leq$  כלי לתכנון הפיתוח

# עקיבות דרישות

- ציון המקור לכל דרישה
- מספור כל דרישה, כך שתוצרים בהמשך יוכלו להתלות בה
- מה החשיבות בכך?

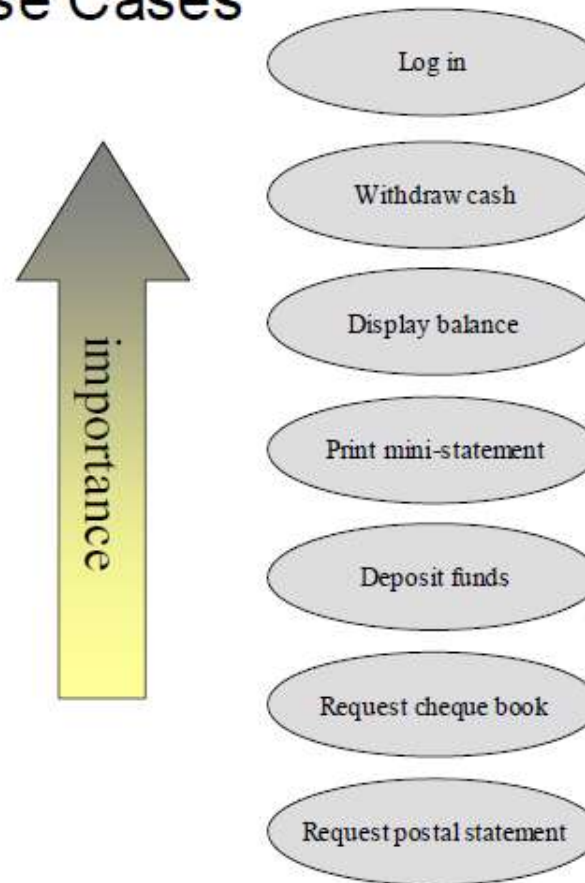


**PRIORITIZE**

At Least You Can Fix The Bike.

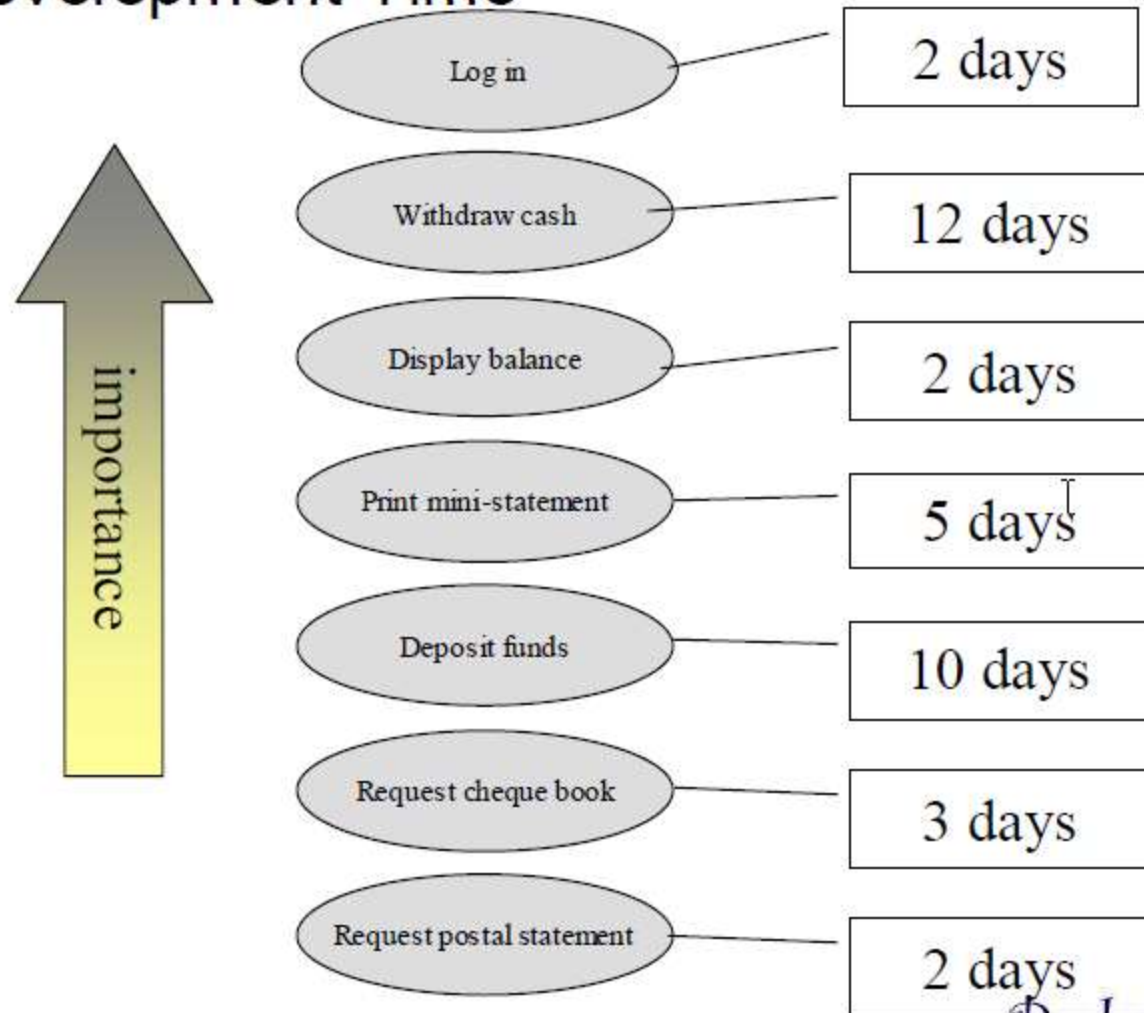
# תיעדוף תרחישים

## Prioritise Use Cases



# הערכה בעזרת תרחישים (בהמשך)

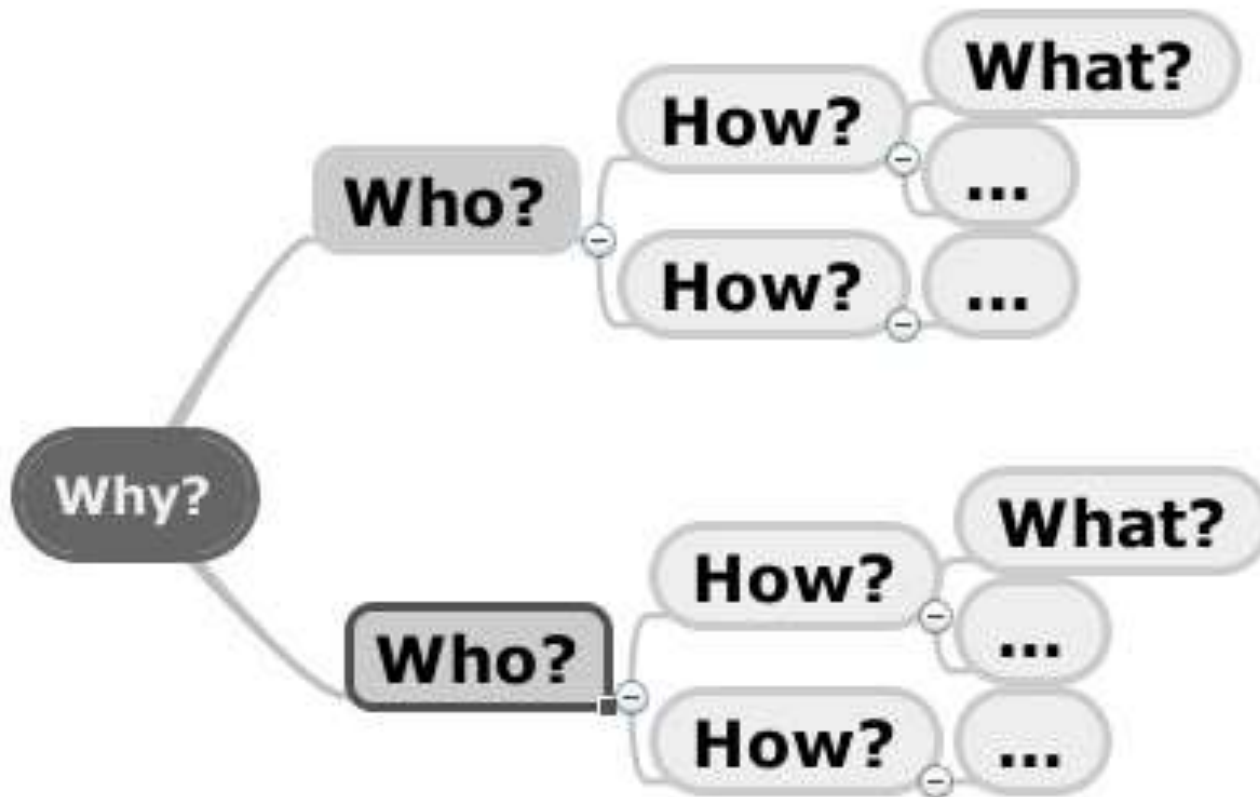
## Estimate Development Time



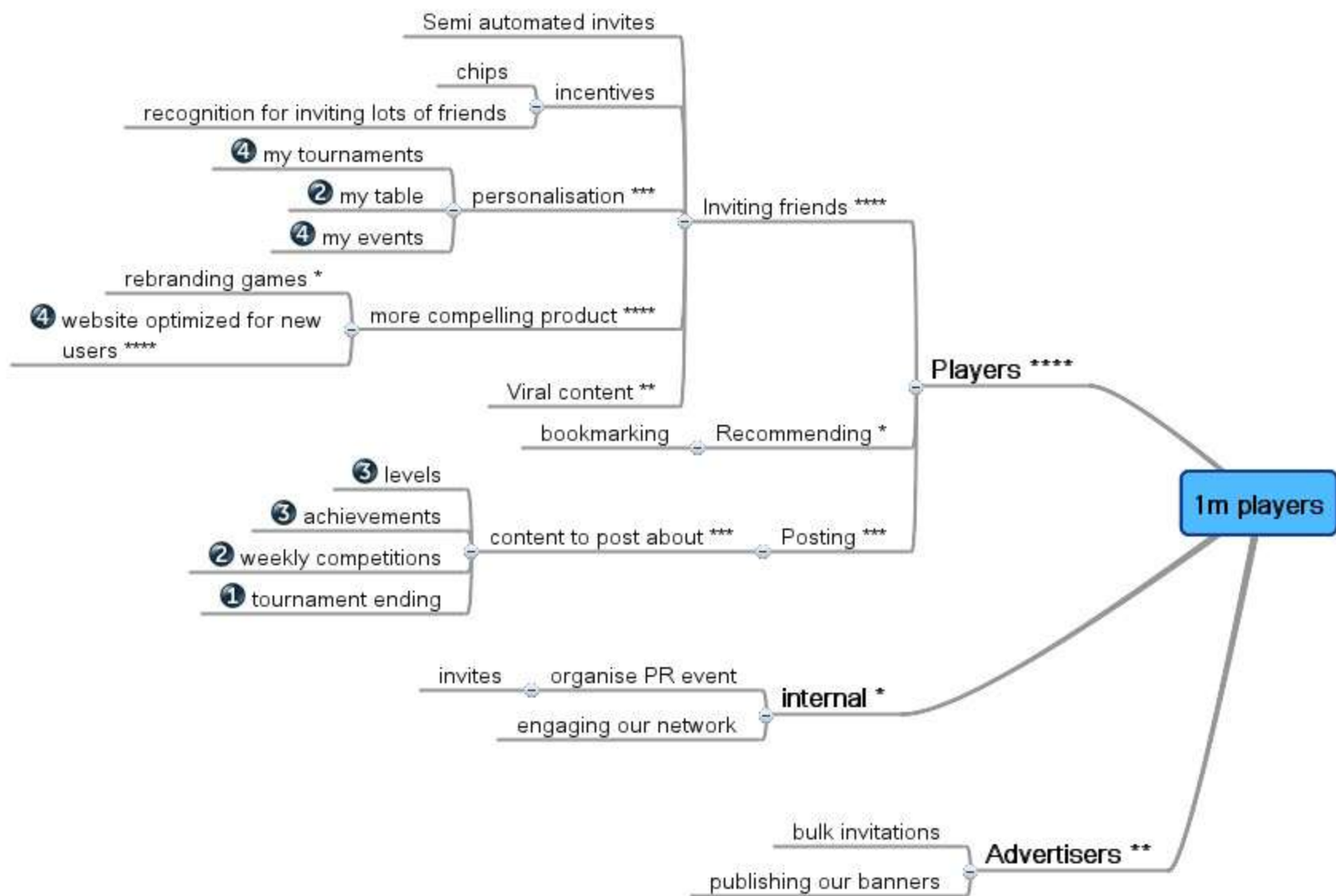
# ניהול ותכנון עם דרישות

- כלים: Doors, [SpecLog](#), GitHub Issues, ...

# Gojko, Agile product management using Effect Maps







# תרגיל

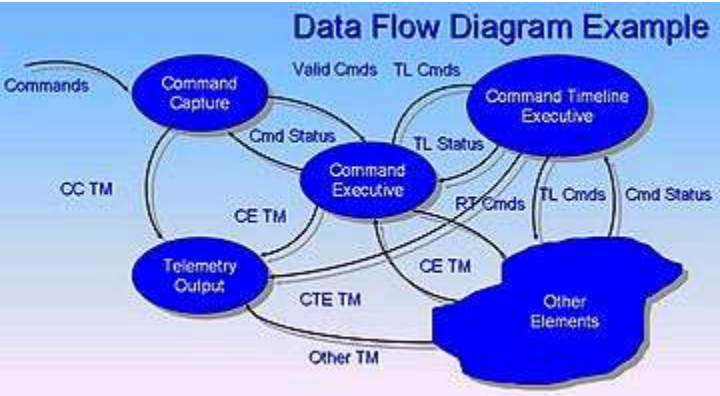
- כל צוות מתחלק לשתי קבוצות
- כל קבוצה עובדת על UC אחד למוצר שלכם



התוצרים יכולים כבר לשמש למסמך  
ה-SRS שלכם

# בפעם הבאה

- סקר מפרט דרישות (רישום באתר הרצאה\תרגיל, דרושים:  
צוותים מתנדבים (בנוס: משוב מוקדם), לקוחות מעורבים)
- דיברנו על ה"מה?"
- בדרך אל ה"איך?" וה"מתי?":
  - תיכון\ארכיטקטורה
  - הערכה ותכנון + כלי מרכזי: בקרת גרסאות
- לקריאה: Booch, [Draw Me a Picture](#)  
(המאמר זמין ממאגר הספרייה ieeexplore digital library  
לחלופין ניתן להאזין לו [כאן](#), שיתוף בפיאצה?)
- שאלה: מהו היתרון המרכזי של דיאגרמות לפיתוח תוכנה?



# סיכום\דיון

- כמה צריך לפרט?
  - פשטות לעומת נכונות מפורטת
  - גראפי לעומת תאור מילולי וטבלאי מדויק
  - קצר ולענין לעומת מלא ומאוחר
- האיזון תלוי גם בלקוח ובטיב ביחסים איתו (או שיטת הפיתוח)
  - האם אחרי אג'יל יש מקום ל-SRS?
  - תרחישי שימוש: אנשים הם גם חלק במערכת!
- פיתוח מפרט בסבבים (הורדת\דחיית מה שלא הכרחי, הפשטה)
- UC ככלי המרכזי ב-OO לנתוח דרישות
  - דיאגרמות נוספות Activity, State Machine
  - בעבר, למשל ב-Structured Analysis
  - דיאגרמות אחרות: Context, ERD, **DFD**
- הנדסת דרישות כמקצוע