"סוף מעשה במחשבה תחילה"

הנדסת תוכנה

4. תיכון Design

Metz: “The purpose of design is to allow you to do design later and its primary goal is to reduce the cost of change.”

מה היום \ השבוע?

- תיכון
  - ~~ארכיטקטורה, תיכון~~, "מרצה אורח"
  - UML \ סדנת תיכון
- פרויקט: כתיבת מפרט תיכון SDS
  - (בתרגיל + השלמת אב טיפוס צד לקוח)
- סקרי מסמך דרישות SRS בכיתה \ Offline
  - השלמת שלב הדרישות
  - סיכום סקר

מקורות – תיכון

- Pressman, chap. 10
- Practical UML: A Hands-On Introduction for Developers
- Ambler, Introduction to Object-Orientation and the UML
- Beck, A Laboratory For Teaching Object-Oriented Thinking, OOPSLA’89 & SIGPLAN Notices 24(10)
- Code Complete, Steve McConnell, Chapter 5
- http://vlib.eitan.ac.il/uml/index.htm חומרים בעברית
- Fowler, UML Distilled

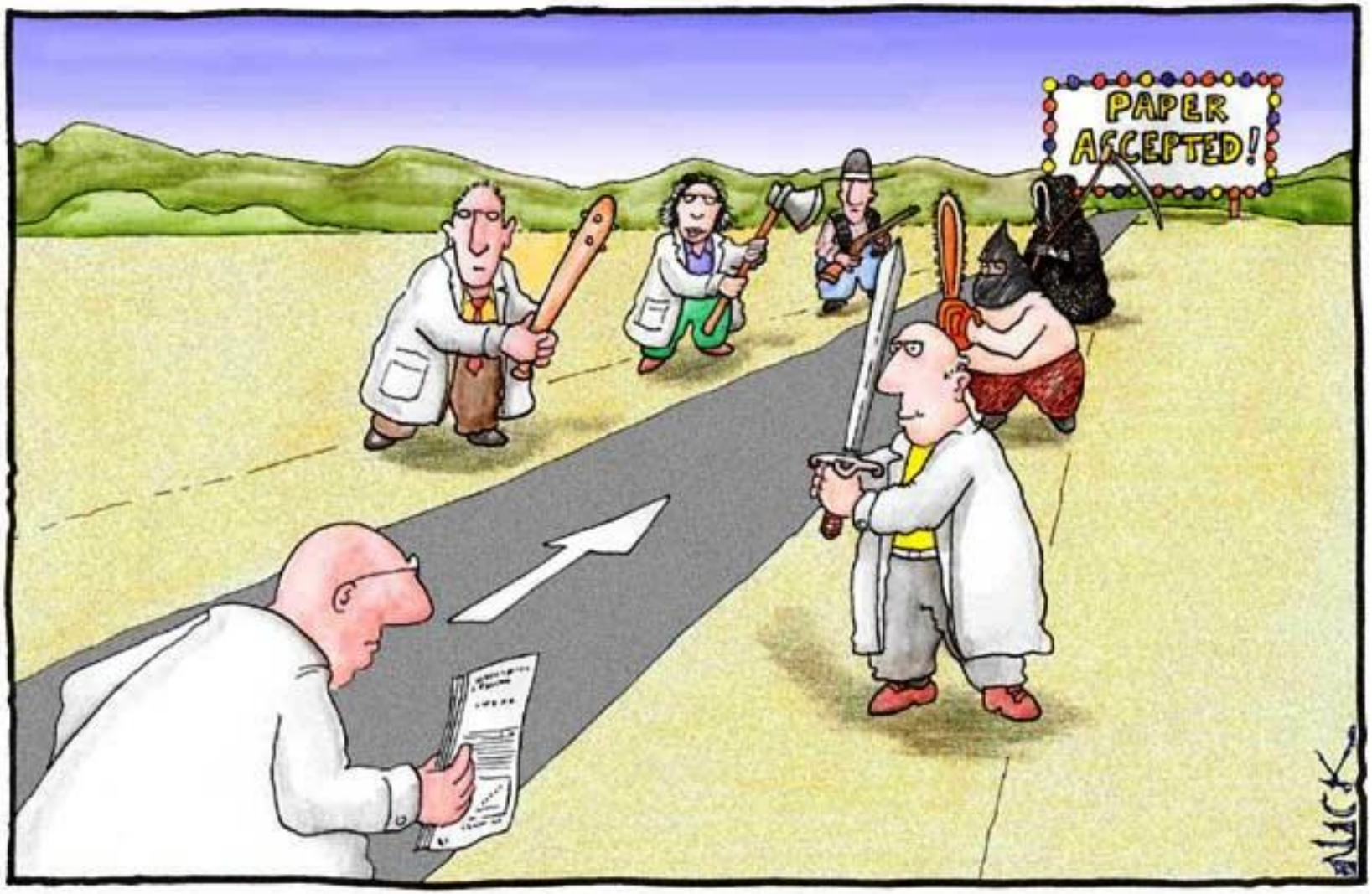
מקורות - ארכיטקטורה

- Pressman, Chap. 9
- Software Project Survival Guide, Chap. 10: Architecture
- Garlan, Software Architecture,
- Booch, The Architecture of Web Applications
- Coplien et. al., Lean Architecture (summary p.5)
- Clements et. al., Documenting Software Architectures: Views and Beyond (2nd Edition)
- Object Oriented Analysis and Design with Applications

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)

מסמכים וסקרים



מסמכים וסקרים

- סקר דרישות SRS Review
  סקר תיכון SDS Review (בפעם הבאה)
- סקרים נוספים מקובלים:
  System Design Review
  Preliminary Design Review
  Critical Design Review
- מסמכים\שמות נוספים (פרויקט גמר):
  User Requirement Document
  Software Design Document
  מסמכי בדיקות ATD/STD
- מה מטרתם? למה כ"כ הרבה (TLAs)? כיצד כדאי לארגן אותם?

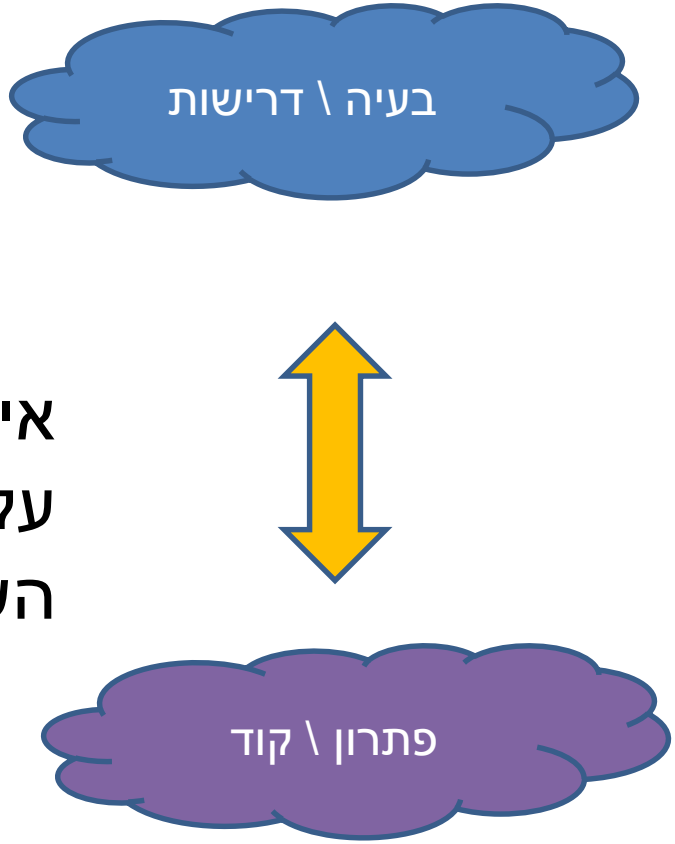
שאלות

- מה הוא תיכון (design)?
- מה היא ארכיטקטורת תוכנה?
- מה תפקידם בתהליך הפיתוח?
- מה תפקידו של ארכיטקט?
- האם עדיין צריך אותם?
  - “Working Software over Comprehensive Documentation”…
  - Just Enough Software Architecture
  - Simon Brown: “a good software architecture enables agility”
- האם שפות וכלים יכולים לעזור?

תיכון Design

- מה זה? למה זה טוב? האם צריך את זה?
- Design (wikipedia): A plan (with more or less detail) for the structure and functions of an artifact, building or system.
- “sufficient information for a craftsman of typical skill to be able the build the artifact without excessive creativity” – Guy Stelle
- “Design is the thinking one does before building” - Richard P. Gabriel
- “design is there to enable you to keep changing the software easily in the long term” - Beck
- פלט משלב זה: SDS – מפרט תיכון תוכנה

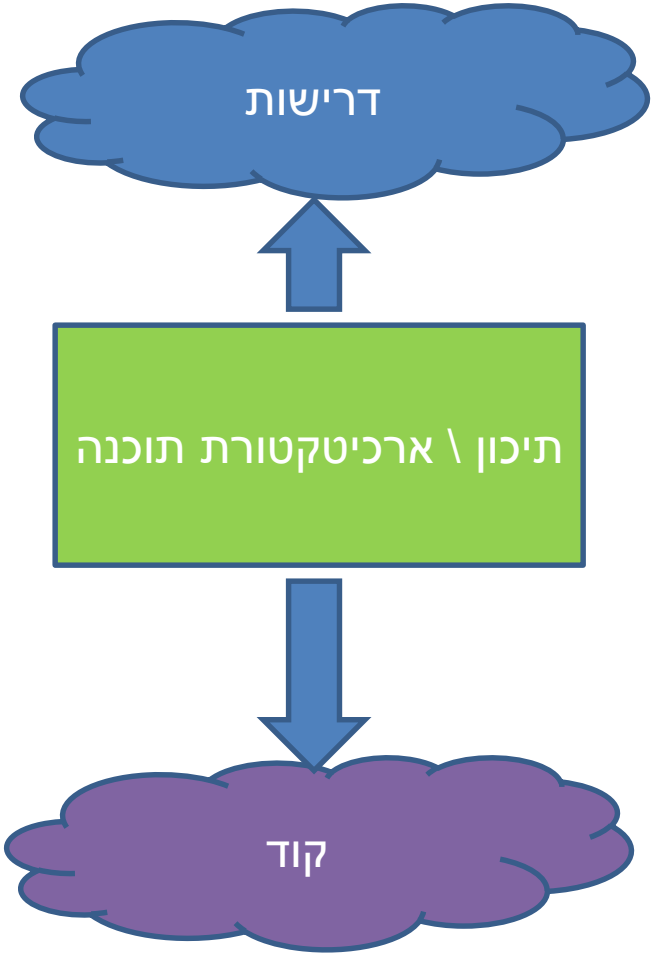
הבעיה העיקרית



איך מגשרים
על הפער בין
השניים?

תשובה ראשונה

תשובה יותר מעשית (היום)



מספקת מסגרת
כללית לבנות
ולפתח את התוכנה

jsilverMDX added a note 21 days ago

I started as a hacker. With .NET, I was a coder. With PHP, I was a programmer. With JavaScript I was a developer. Then I came to Ruby and experienced great tools like Rake and Gem, and became a Software Engineer. Thank you Jim Weirich, you have helped me immeasurably in my career. Ruby, nor me, would be where they are today without you. You have set a very high standard to be strived for in the next generation of software engineering teachers and collaborators. Rest in peace.

"מרצה אורח"

- Jim Weirich RIP ☹, OSS, Lecturer (future)
- Good: Legend, Succinct (for SDS), Diversity
- Bad (but): English, Ruby (Rails), non-students
- Ugly: rest of lecture? How was it?
- Last: "The Big Picture" (UML for geeks)
- Miami beach 0:30 big picture 6:00 uml class diagram 18:00 rails arch.+ adv. testing discussion 24:00 rake example 28:00, object diagram 30:40 flying robots example 38:10 dynamic behavior /state diagram 39:35 mock lib. example sequence diagram 44:15 summary, tips, tools

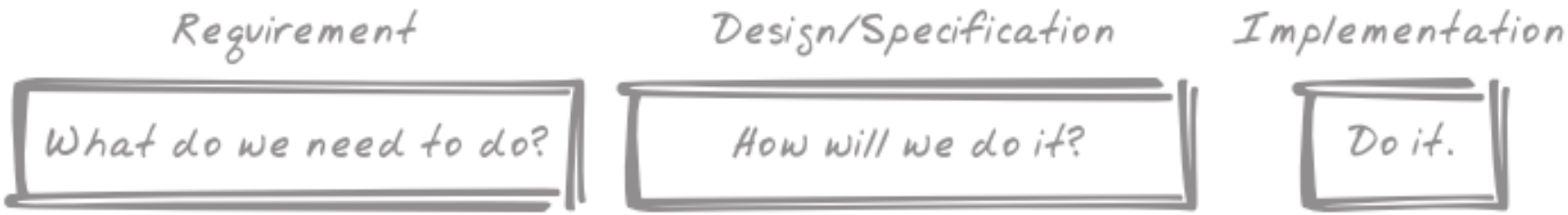
עבור מה כדאי להשתמש בשפת UML?

1. לתקשר את התיכון שחשבנו עליו
2. תבניות ליצירת קוד (Code Generation)
3. שפת תוכנה ויזואלית (עוד דוגמא LabView)
4. מתכנתי רובי ו-web לא משתמשים ב- UML

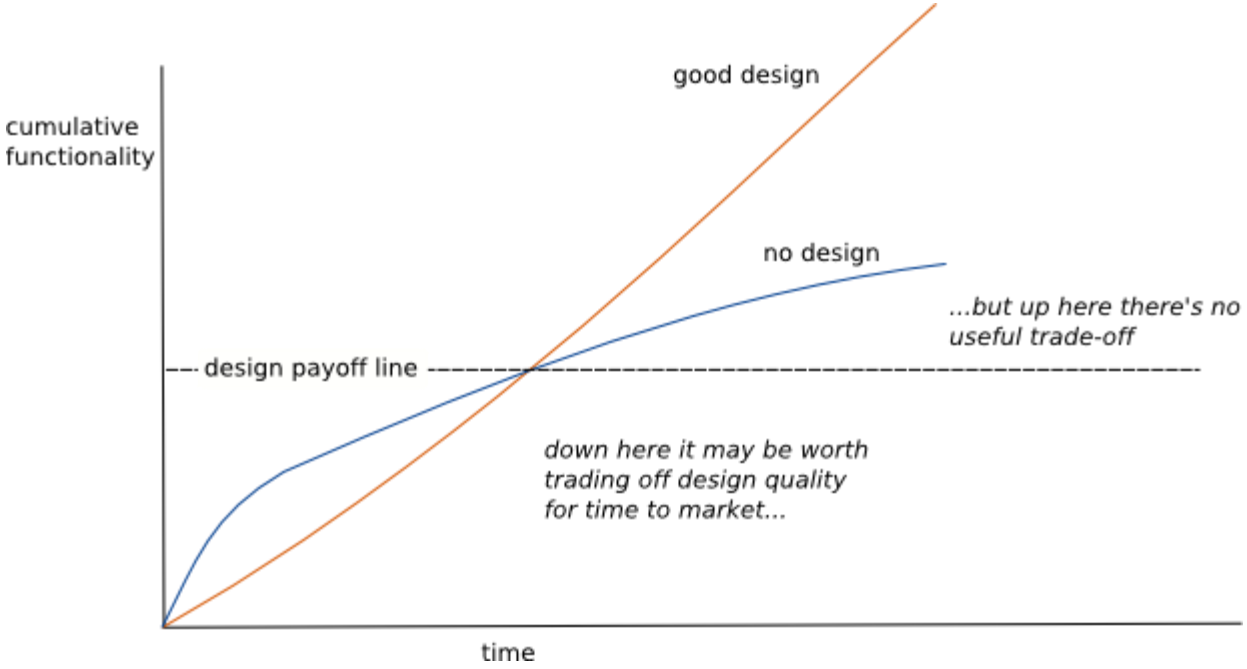
מה עוד

- מדדים לתיכון טוב
- תיכון פשוט (אג'ייל)
- CRC – סדנה לזיהוי וחלוקה למחלקות
- מקורות ל- UML
- תרגיל: יומן
- סיכום
- משימת SDS

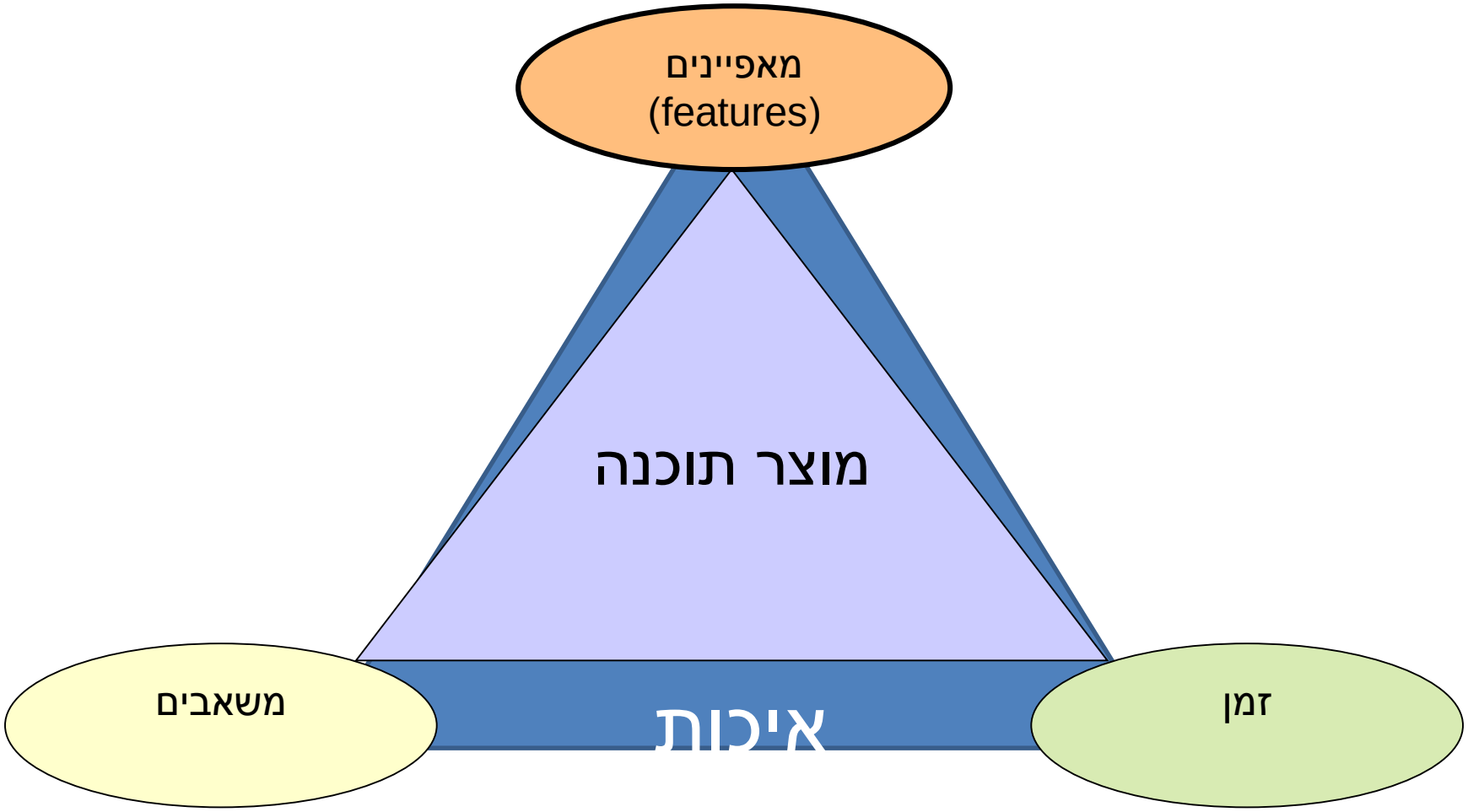
חלק בתהליך


Requirement
What do we need to do?
Design/Specification
How will we do it?
Implementation
Do it.

האם כדאי להשקיע בתיכון?



תזכורת: פרויקט תוכנה:



ארכיטקטורה

“Good software architecture makes the rest of the project easy”
McConnell, Survival Guide

Booch: Traits of Successful Projects

- A successful software project is one in which the deliverables satisfy and possibly exceed the customer expectations, the development occurred in a timely and economical fashion, and the result is resilient to change and adaption.
- … several traits that are common to virtually all successful oo systems we have encountered and noticeable absent from the ones we count as failures:
  - Existence of a strong architectural vision
  - Application of a well managed iterative and incremental development lifecycle

ארכיטקטורה: הגדרה

- ארכיטקטורה של מערכת מתארת את המבנה העיקרי שלה, כך שהיא תתאים לצורכי הלקוח תוך כדי עמידה באילוצי טכנולוגיה ותקציב
  - המרכיבים העיקריים וההתנהגות שלהם
  - הקשרים בין מרכיבים אלו
  [מתוך מכתב של Brooks]

ייצוג היבטים שונים של התוכנה באופן מופשט

Yegge About Bezos

- His Big Mandate went something along these lines:

  1) All teams will henceforth expose their data and functionality through service interfaces.

  2) Teams must communicate with each other through these interfaces.

  3) There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

  4) It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols -- doesn't matter. Bezos doesn't care.

  5) All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.

  6) Anyone who doesn't do this will be fired.

  7) Thank you; have a nice day!

  Ha, ha! You 150-odd ex-Amazon folks here will of course realize immediately that #7 was a little joke I threw in, because Bezos most definitely does not give a shit about your day.

עוד הגדרות

- Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change (Booch 2006)
- The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment and the principles guiding its design and evolution (IEEE1471 2007)
- The form of a system (Coplien, Lean Arch. 2010)
- “things that people perceive as hard to change” (Fowler, Who Needs an Architect, 2003)
- “the set of design decision which, if made incorrectly, may cause your project to be cancelled” - Eoin Woods (SEI 2010)
- “In a sense we get the architecture without really trying. All the decisions in the context of the other decisions simply gel into an architecture” – Cunningham, 2004
- "irreversible decisions in the large“ J. B. Rainsberger
- Many more @ SEI Community Software Architecture Definitions

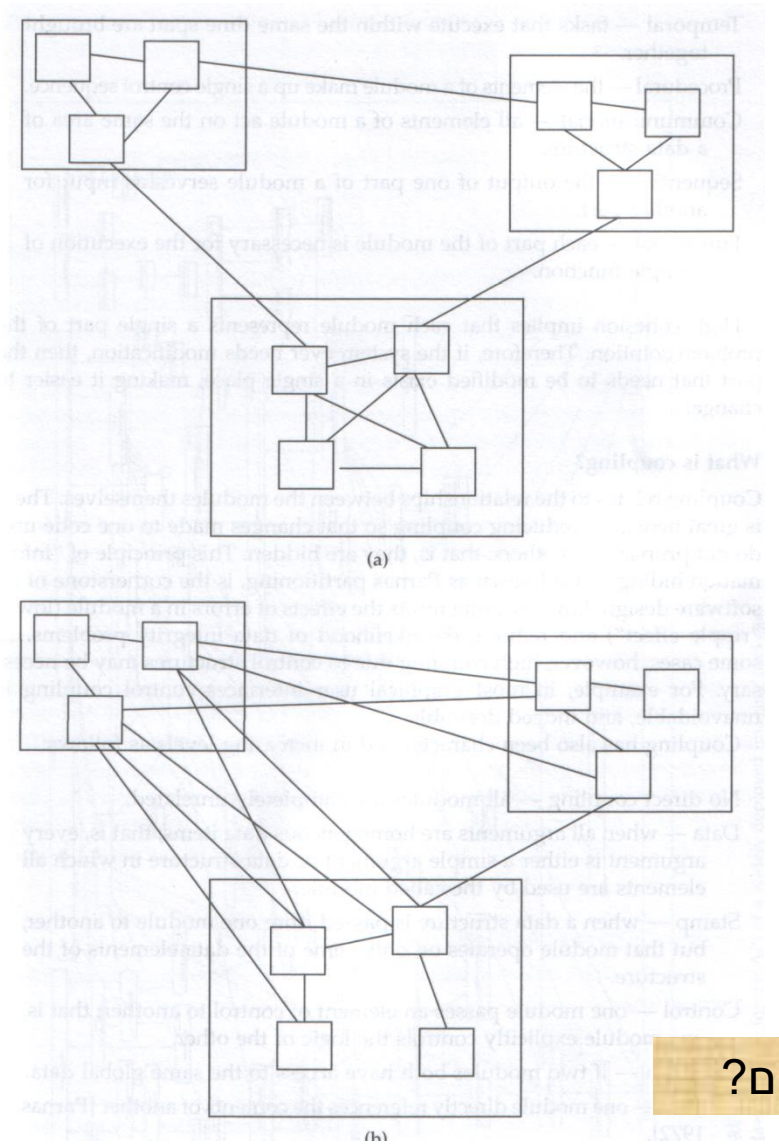
- http://en.wikipedia.org/wiki/Software_architecture
  'To date there is still no agreement on the precise definition of the term “software architecture”. However, this does not mean that individuals do not have their own definition of what software architecture is. This leads to problems because many people are using the same terms to describe differing ideas.'

מאפייני ארכיטקטורה טובה

- חלוקה לשכבות ברורות המייצגות כל אחת הפשטה* של המערכת ומספקת ממשק ברור
- הפרדה בין הממשק והמימוש של כל שכבה
- פשטות: שימוש בהפשטות* ומנגנונים מקובלים

* Uncle Bob Martin: “Abstraction is the elimination of the irrelevant and the amplification of the essential.“

מדדים מרכזיים לתיכון תוכנה



- אנליזה ומודולריות: חלוקה לרכיבים והקשרים בינהם
- Coupling – צימוד
  - מדד תלות באחרים
- Cohesion – לכידות
  - מדד ענייניות
- עקרונות מפורטים יותר בהמשך

האם כדאי שהמדדים יהיו גבוהים או נמוכים?

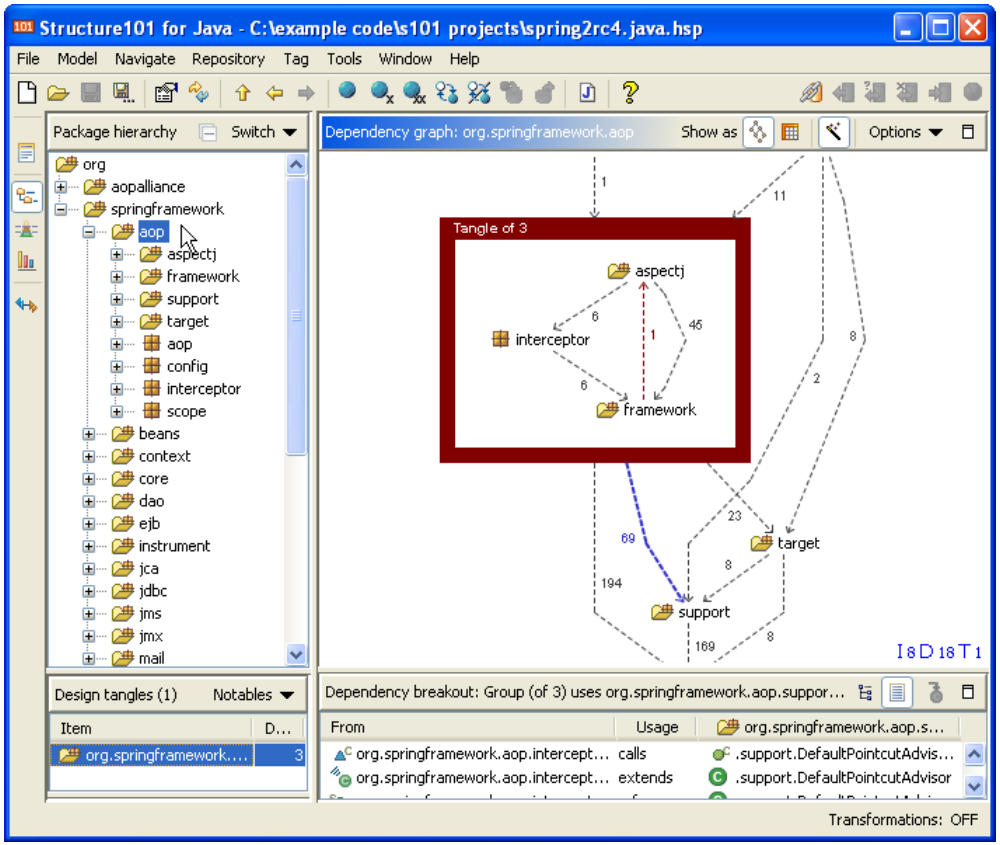
Cohesion Types [Yourdon]

7.	Informational cohesion	(Good)
6.	Functional cohesion	
5.	Communicational cohesion	
4.	Procedural cohesion	
3.	Temporal cohesion	
2.	Logical cohesion	
1.	Coincidental cohesion	(Bad)

http://en.wikipedia.org/wiki/Cohesion_(computer_science)#Types_of_cohesion
http://highered.mcgraw-hill.com/sites/dl/free/0073191264/371536/Ch07.pdf
http://robots.thoughtbot.com/post/23112388518/types-of-coupling

מדדים וכלים

- מדדי סיבוכיות שונים (דוג' Todo MVC)
- כלים, למשל: Structure101 ,Ndpend ruby metric_fu Codeclimate



האם מהנדס תוכנה מתחיל יכול פשוט להשתמש בעקרונות ובמדדים שראינו ולייצר תוכנה טובה יותר?

1. בתאוריה כן אבל למעשה צריך ללמוד להתאים כל עקרון למציאות משתנה
2. בהחלט, אם הוא יודע להחיל אותם במקרים הנכונים
3. העקרונות האלו תאורטיים בלבד ולא נראה לי שיש בהם שימוש בפועל
4. תלוי בשפת התוכנה, יש כאלו שכבר מכילות בתוכן עקרונות אלו

ארכיטקטורה עוזרת ב:

- הבנת המערכת – תאור הקשרים בין מרכיבים
- שימוש חוזר – לאור החלוקה הכללית לרכיבים, זיהוי הזדמנויות
- טיפול בדרישות לא-פונקציונליות (אילוצים)
- מימוש – חלוקה למשימות (במיוחד בצוותים גדולים), וכך נעבור מדרישות למימוש
- ניהול – עוזרת להבין את כמות העבודה ולעקוב אחרי התקדמות
- תקשורת – מייצרת הבנה ואוצר מילים, "תמונה אחת שווה אלף מילים"
  - (אבל ב- Agile: שורת קוד אחת שווה אלף תמונות :-)
- אמורה לאפשר שינוי!

ארכיטקטורה ואילוצי מערכת

- למשל, ביצועים גבוהים:

חלוקה לרכיבים מקביליים, זיהוי צווארי בקבוק, ניהול תקשורת בין רכיבים; קצבים

- אבטחה:

לאלו חלקים מותר לגשת, זיהוי מקומות לשמירה נוספת, הוספת רכיבים שאפשר לסמוך עליהם

- גמישות לשינויים, הרחבתיות:

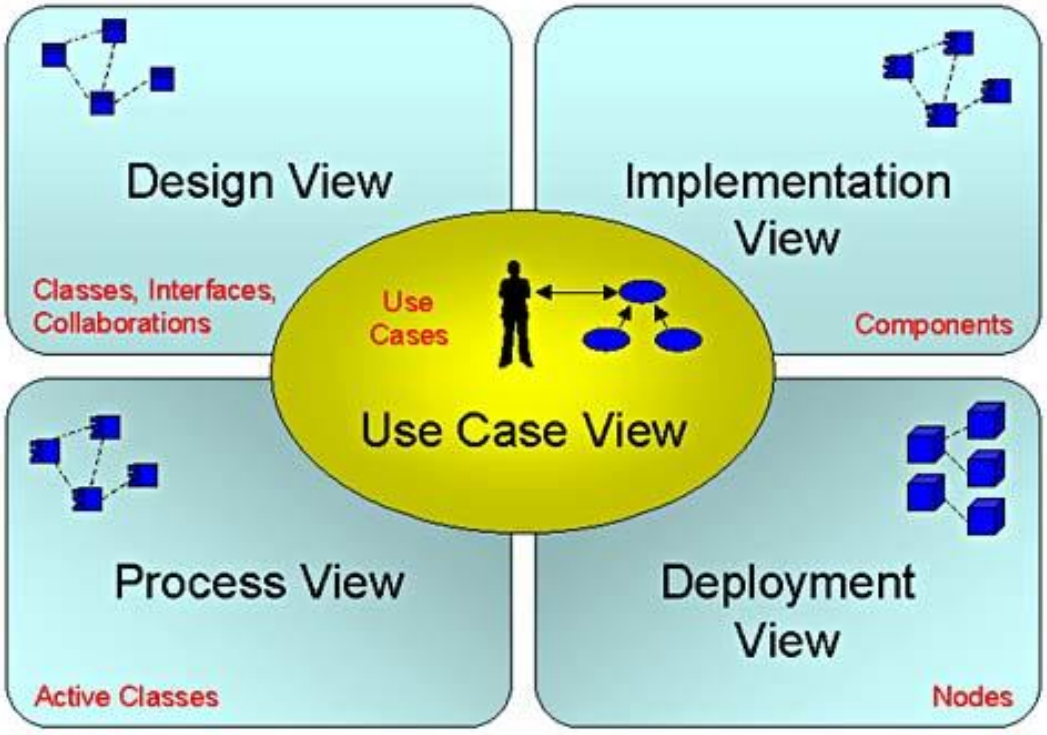
הפרדת רשויות בין החלקים כך ששינויים לא יחלחלו לכלל המערכת

בעצם ישנם היבטים שונים

- "היבט" מאיר ומדגיש אוסף של החלטות ראשיות בתכנונה של מערכת
  - כיצד תורכב המערכת מחלקים שונים
  - היכן הממשקים העיקריים בין החלקים
  - מאפיינים עיקריים של החלקים
  - מידע שמאפשר המשך ניתוח והערכות

החשיבות של היבטים (Views)

- היבטים שונים נדרשים, כדי להבין ממדים שונים של המערכת (Philippe Kruchten)



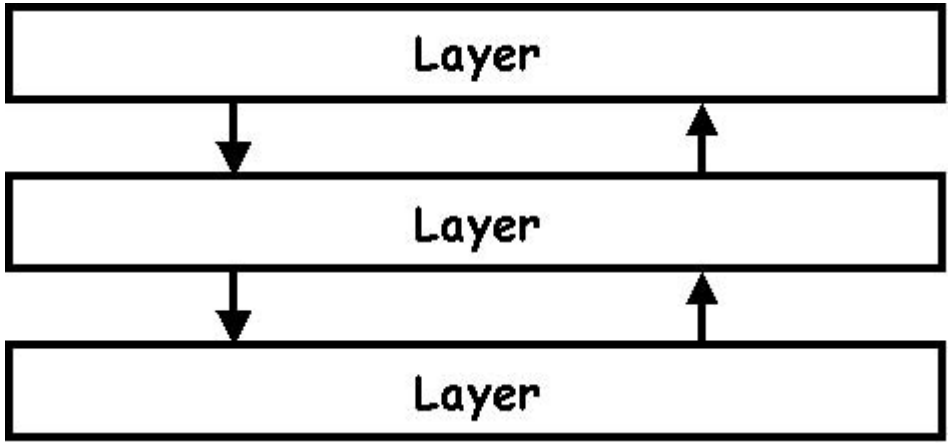
Booch

כמה דוגמאות לארכיטקטורה

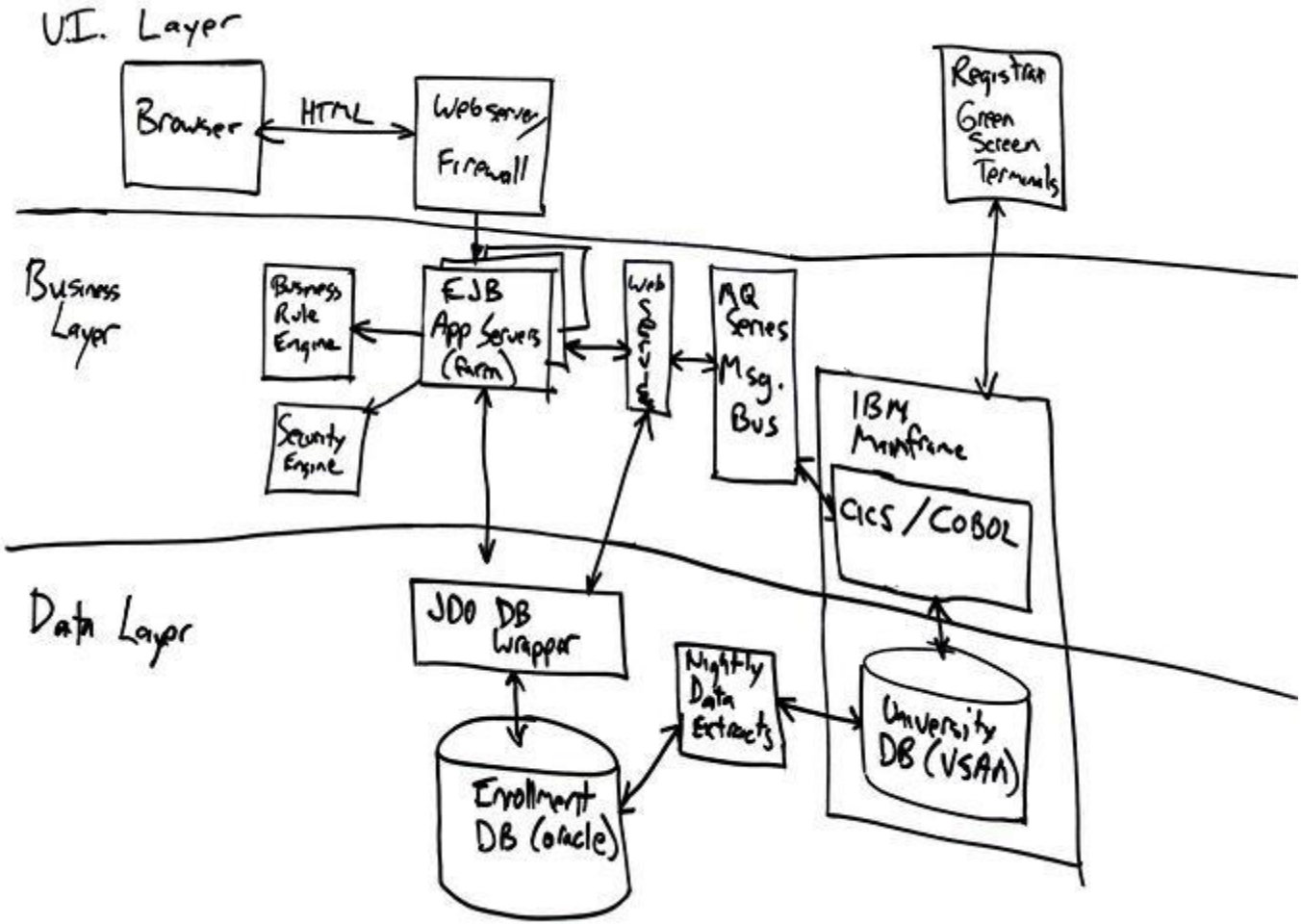
- תבניות שימושיות שהתפתחו מתוך נסיון
- מתאימות בהקשר מסוים
- בסיס להתאמה לבעיה הנוכחית
- בהמשך: תבניות תיכון\עיצוב Design Patterns

"Architecture should reflect use cases not a framework" - R. Martin

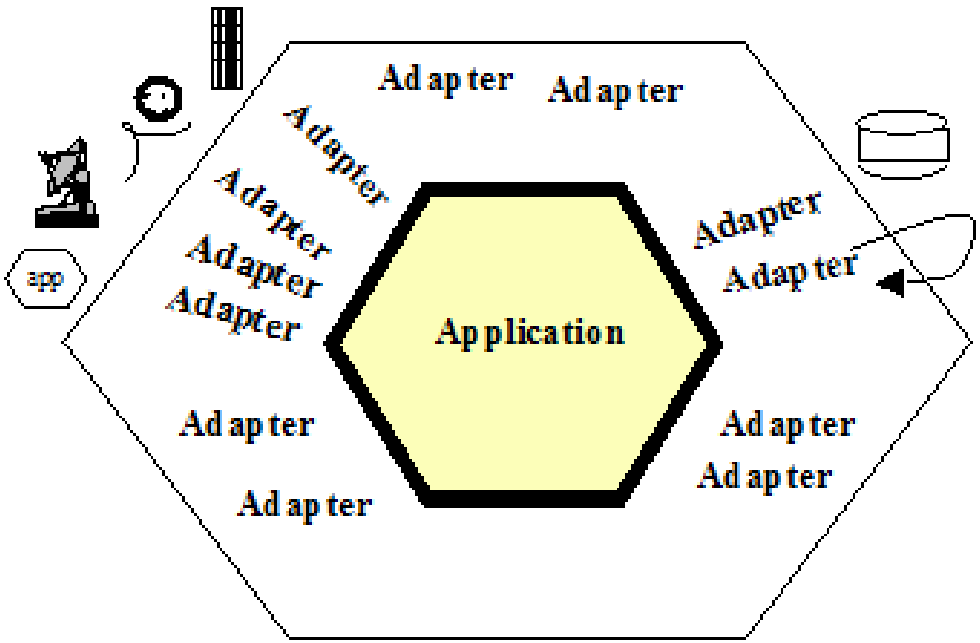
שכבות



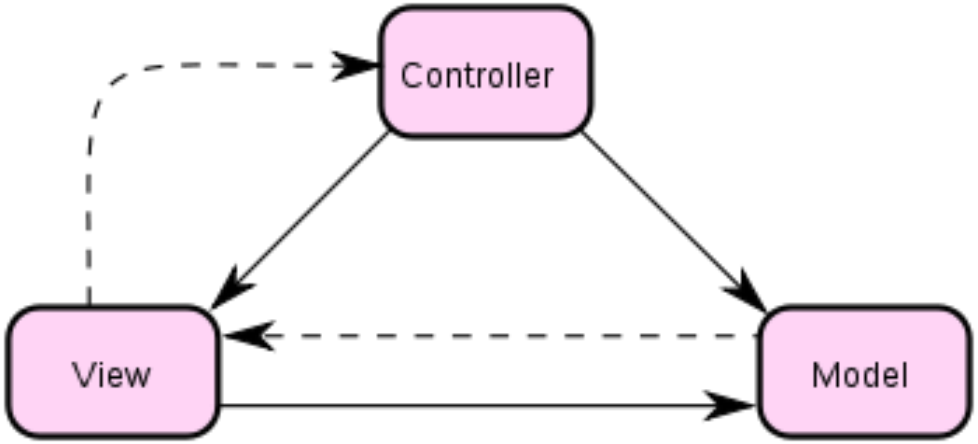
- דוגמאות: מערכות הפעלה, פרוטוקולי תקשורת, N-tier


U.I. Layer
Browser
HTML
Web Server
Firewall
Registrar
Green
Screen
Terminals
Business
Layer
Business
Rule
Engine
EJB
App Servers
(farm)
Web
Services
MQ
Series
Msg.
Bus
IBM
Mainframe
CICS / COBOL
Security
Engine
Data Layer
JDO DB
Wrapper
Nightly
Data
Extracts
University
DB (VSAM)
Enrollment
DB (oracle)

Hexagonal Architecture / Ports & Adapters



Model View Controller



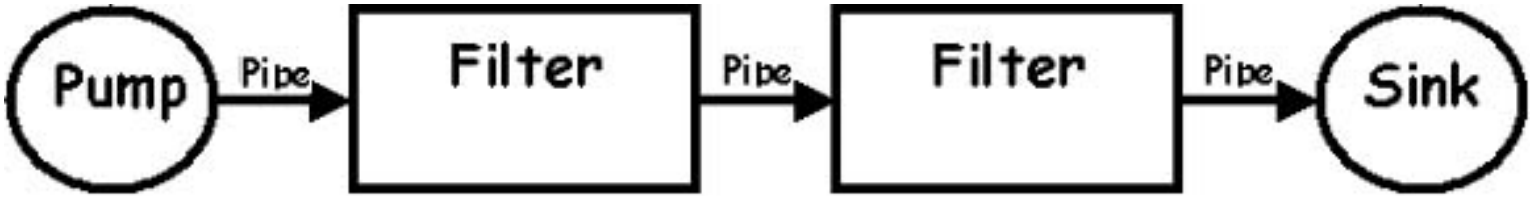
- מימושים עכשוויים:

Ruby on Rails, ASP.Net MVC, Angular.js

REST

- ארכיטקטורה לבניית מערכות מבוזרות
- אילוצים
  - הפרדה ללקוח ושרת
  - אין שמירת מצב לקוח
  - שימוש במטמון
  - שכבות
- => מדרגיות, הפצה נוחה ועוד
- דוגמא: שכתוב טוויטר לעמידה בצמיחה

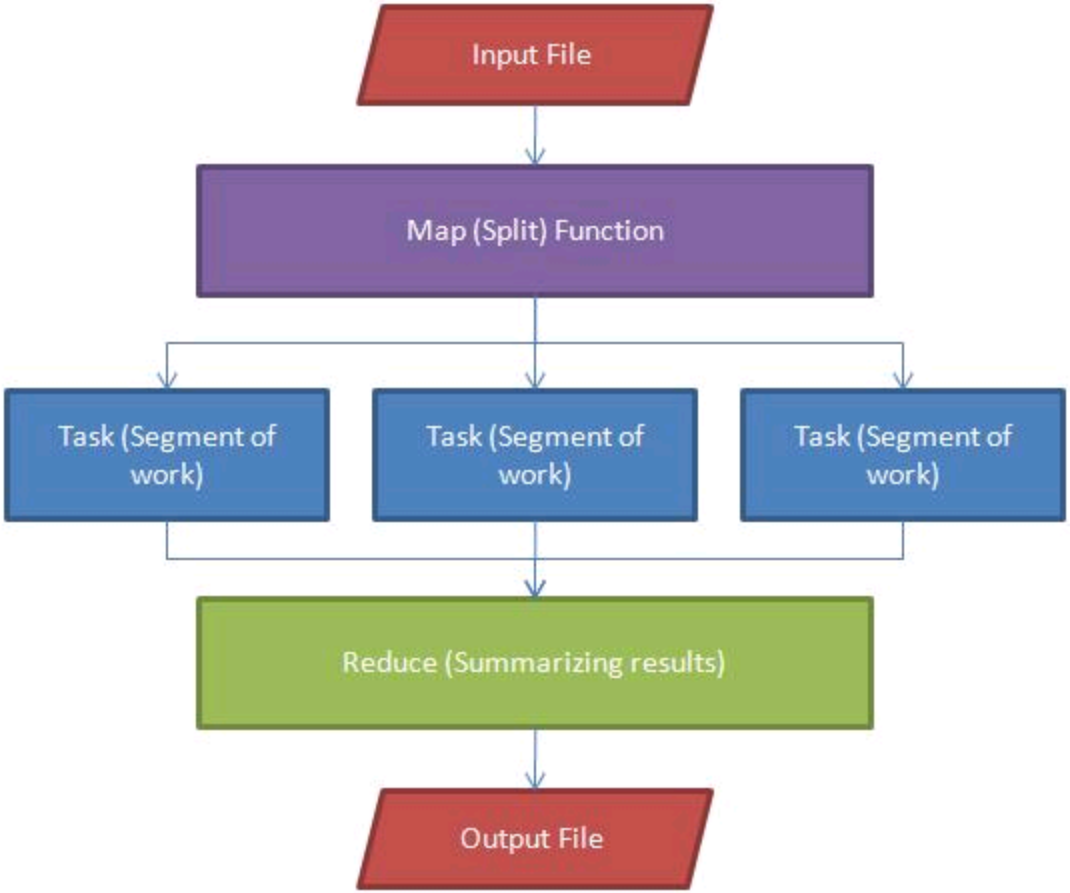
Pipe and filter



דוגמאות:

- Unix: ps aux | grep init
- מהדרים

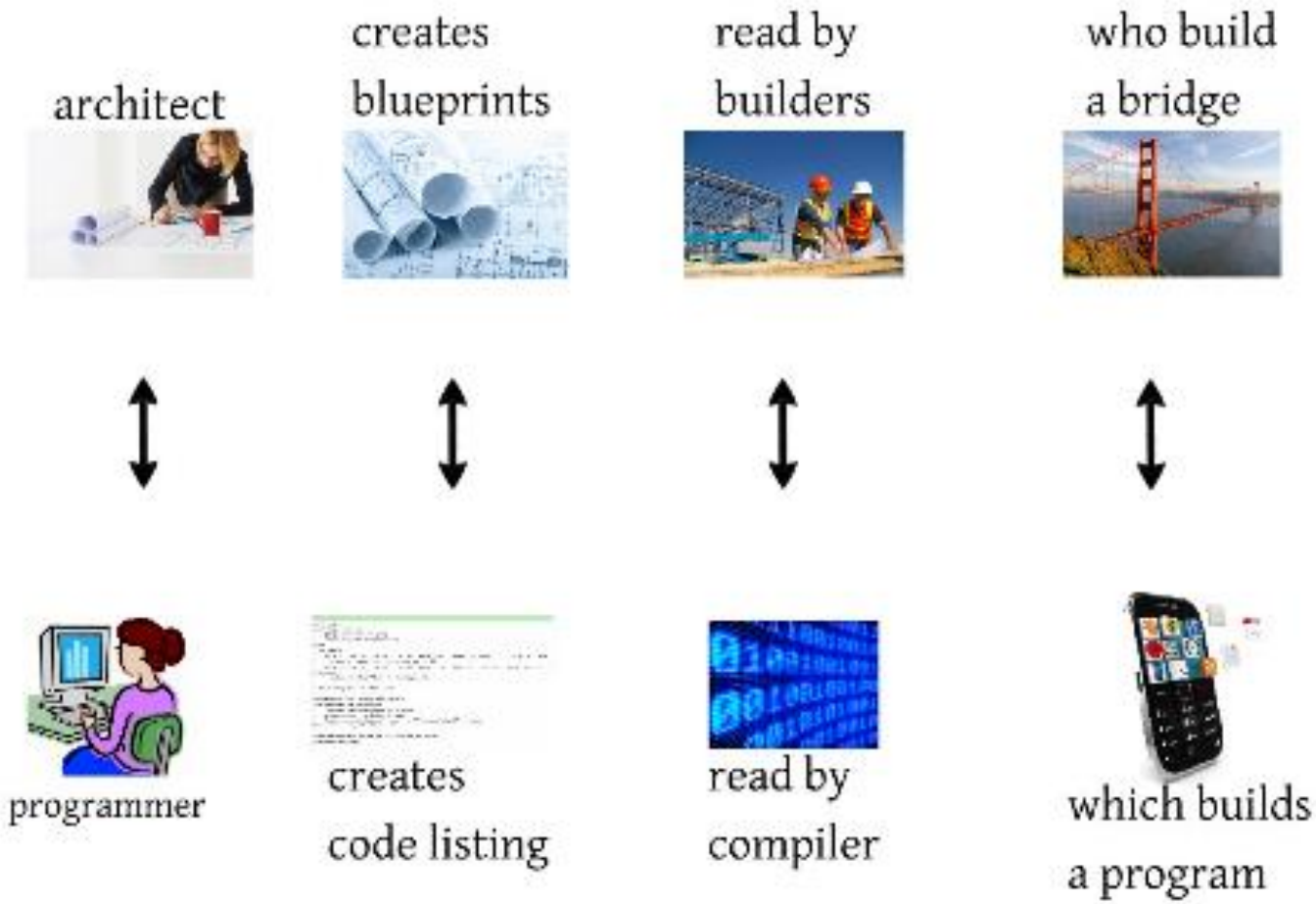
Map-Reduce (Google)



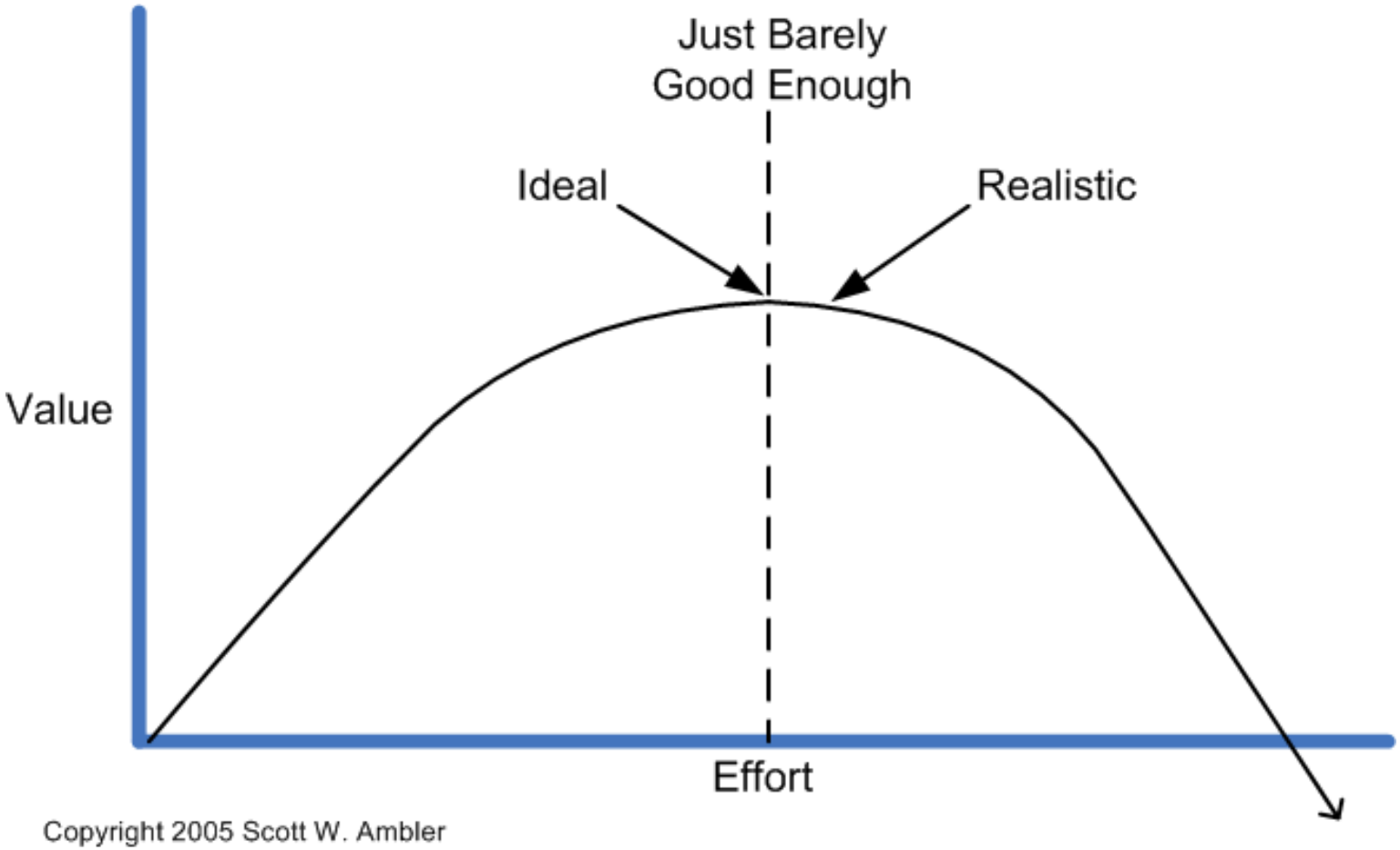
דיון

- מה ההבדל בין ארכיטקטורת תוכנה לכל ארכיטקטורה אחרת (מה בא אח"כ)?
- מה בעצם ההבדל בין ארכיטקטורת תוכנה לתיכון (Design)? האם יש קשר לעיצוב?
- האם אפשר להתחיל לקודד ישר מהדיאגרמות שראינו?
- אם מפתח בחר את ה-Model והשותף את ה-View, האם אפשר ללכת לפתח ולהפגש עוד חודש לאינטגרציה

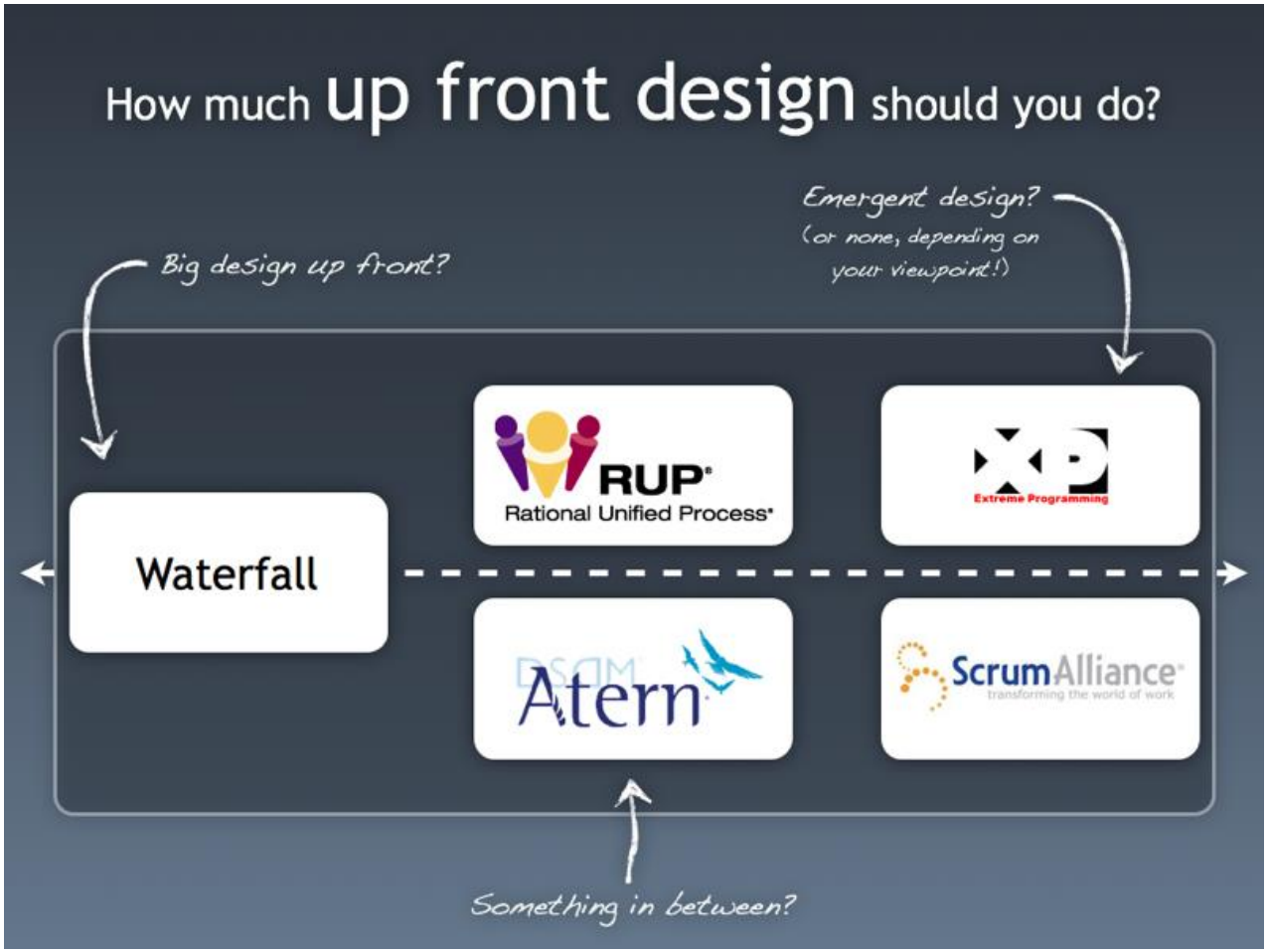
Reeves, The Code is the Design!



Good enough כמה צריך?



כמה צריך לתכנן מראש?



Beck (XP): Simple Design

1. כל הבדיקות עוברות
2. ללא כפילויות (DRY)
3. ברור - מבטא את כוונת המתכנת
4. קטן - מינימום של מחלקות ומתודות

- אפשר פחות?
  The Four Elements of Simple Design
- Fowler, Is Design Dead?

זיהיתי קוד כפול אך הוצאתו למחלקה נפרדת עלולה להפוך אותו לפחות קריא - מה לעשות?

1. תמיד נעדיף למנוע כפילויות
2. קריאות הקוד חשובה יותר
3. קריאות חשובה יותר בתנאי שלא נגדיל את מספר המחלקות
4. קודם נכתוב בדיקה למחלקה החדשה ואם היא תעבור נוציא את הקוד

האם באמת אפשר לוותר על תיכון?
Balance Design and Refactoring

EXTREME PROGRAMMING

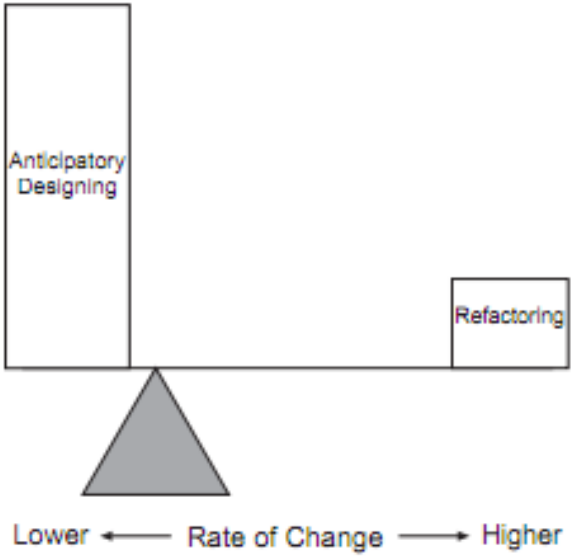


Figure 3 — Balancing design and refactoring, pre-internet.

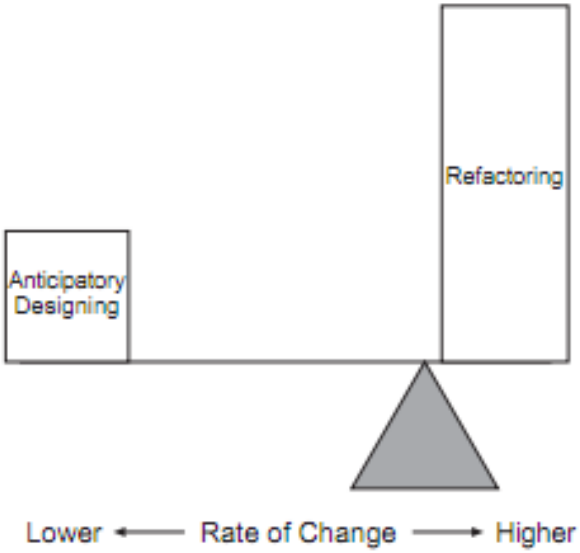


Figure 4 — Balancing design and refactoring today.

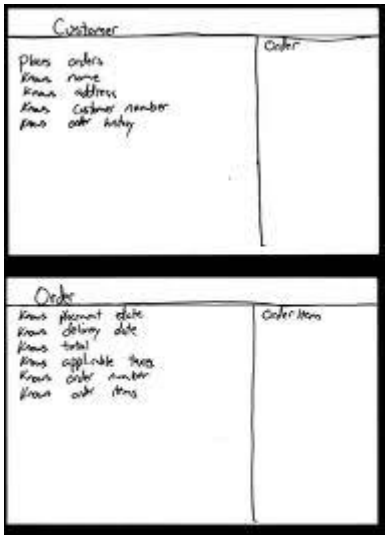
תיכון מעבר לארכיטקטורה

- תיכון מפורט
- תיכון שאינו ארכיטקטוני
- תיכון פונקציונלי

- כללי אצבע ועקרונות תיכון (מונחה עצמים – בהמשך) למשל:

Don’t Repeat Yourself, Tell Don’t Ask, Law of Demeter, Favor Composition Over Inheritance, **S**ingle Responsibility Principle (SOLID), …

כיצד באים אובייקטים לעולם?



- חילוץ מהדרישות
- כרטיסי CRC
- תרשימי UML
  - עוד בהמשך (ובתיכון מונחה עצמים)
- ועוד (למשל Domain Driven Design - Evans)
- Parnas, D. On the Criteria To Be Used in Decomposing Systems into Modules., `72

it
b
o
b
w
b
d
w
m
s

Class Responsibility Collaboration Cards

- הוצע ע"י Beck & Cunnigham ב-1989
  - A Laboratory for Teaching Object-Oriented Thinking
  - המטרה: לחשוב מראש על מערכת כאוסף אובייקטים במקום תכנות פרוצדורלי
- התהליך: לכל תרחיש או סיפור\ים:
  - זיהוי מחלקות
  - זיהוי אחריות לכל אחת – דברים שמבצעת \ יודעת
  - זיהוי שותפים העוזרים לה להגיע למטרתה
  - שינוי ועדכון תוך כדי עבודה על תרחישים נוספים

דוגמת CRC

Showing	
Responsibilities	*Collaborators*
Knows name of movie	Movie
Knows date & time	
Computes ticket availability	Ticket

Ticket	
Responsibilities	*Collaborators*
Knows its price	
Knows which showing it's for	Showing
Computes ticket availability	
Knows its owner	Patron

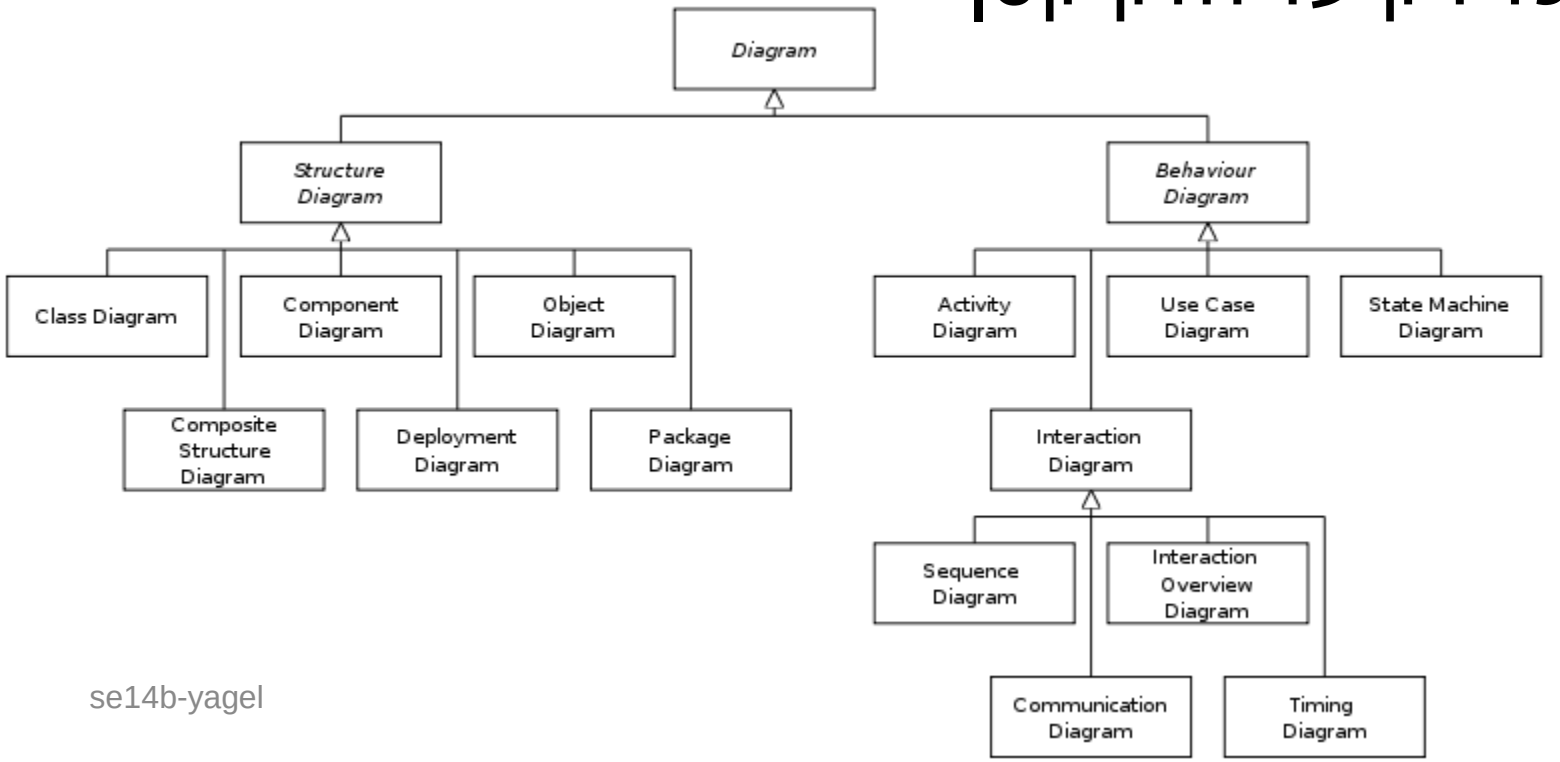
Order	
Responsibilities	*Collaborators*
Knows how many tickets it has	Ticket
Computes its price	
Knows its owner	Patron
Knows its owner	Patron

תיכון עם תרשימים - UML

- "שפה" מקובלת UML
  - הרחבות לארכ' \ מערכות למשל SysML, AADL
- האם באמת מקובל? מי מכין תרשימים? האם צריך תפקיד מיוחד? למה הם משמשים?
- כלים (ר' הרצאה קודמת)
- חוויות של סטודנט בתעשייה (רברסים ב10:55)
- Dijksatra: “Every time someone draws a picture to explain a program it is a sign that something is not understood”

UML בקצרה

- Unified Modeling Language – UML
- שפה ויזואלית לתאור ארכיטקטורה, תיכון OO, ...
- נסתכל רק על חלק קטן



Design with UML – Crash Course

- Deployment Diagram (from AgileModeling)
  - Tools, e.g., Visio, StarUML, (web: draw.io, websequencediagrams.com, webwhiteboard.com)
- Class Diagrams (*)
- Sequence Diagrams (*)

AgileData.org: … all developers should have a basic understanding of the industry-standard Unified Modeling Language (UML). A good starting point is to understand what I consider to be the core UML diagrams – use case diagrams, sequence diagrams, and class diagrams – although as I argued in An Introduction to Agile Modeling and Agile Documentation you must be willing to learn more models over time.

תרגיל יומן

- (תרחיש שימוש \ סיפור משתמש)
- הפצה
- CRC
- מחלקות וקשרים
- תרשימי רצף

תרגיל יומן

עוד דיון

- מה בתרשימים מהווה ארכיטקטורה ומה לא?
- האם תיכון מקדם\מאפשר שינויים?
- איך מחליטים מה להכניס לתרשימים?
  - איזו רמת פירוט\אורך? האם מספיקות סקיצות?
  - האם כשהקוד משתנה צריך לעדכן בחזרה?
  - האם להשתמש ב- Code Generation?
  - איפה שומרים אותם? מה התפקיד של התרשימים בכלל?
- מה הקשר בין תיכון לבדיקות?

בפעם הבאה

- סקר SDS – גם מלקוח טכני! (משימה אישית 2 – בטא!)
  (לצפייה בנושא משוב: Feedback without frustration, Berkun)
- לקראת מימוש
  - תהליך ושיטות: הערכה ותכנון
  - כלים: ניהול משימות ותחילת בקרת קוד (VCS)

לסיכום

"While code is the truth, it is not the whole truth"
- Grady Booch

- תיכון ומקומו בתהליך
  - אין צורך להגיע לשלמות אלא לאפשר ...
- ארכיטקטורה \תיכון אחר
  - בהמשך: עקרונות תיכון מונחה עצמים ו- Design Patterns
- UML, CRC, ...
- SDS מפרט תיכון תוכנה (קבוצתי) וסקר (אישי)