



# Software Engineering I

Dr. Elham Mahmoudzadeh  
Isfahan University of Technology  
[mahmoudzadeh@iut.ac.ir](mailto:mahmoudzadeh@iut.ac.ir)

202\

The background features a light gray gradient with several realistic water droplets of varying sizes scattered across the surface. In the center, there is a faint, circular logo. The logo consists of a gear-like outer ring with Persian text 'دانشگاه صنعتی اصفهان' (University of Technology, Isfahan) written along its top arc. Inside the gear is a stylized sunburst or star-like emblem.

# **Chapter 7**

## **Moving To Design(II)**

# Steps(I)

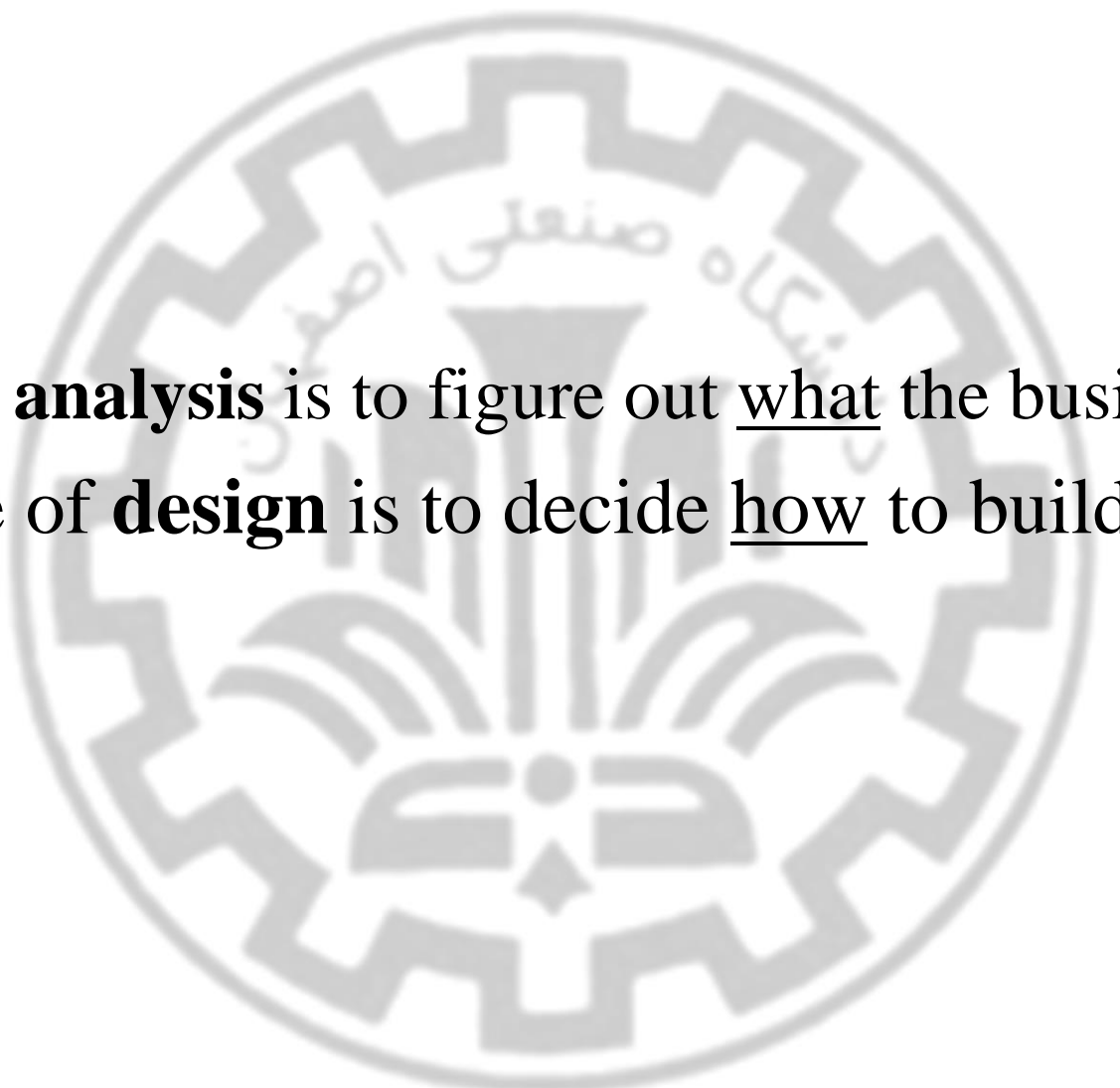
1. Preparing proposal
2. Requirements determination
  - User story
3. Abstract Business Process Modelling
4. Analysis
  - Functional Modelling
  - Structural Modelling
  - Behavioral Modelling

## Steps(II)

### 5. Design

- Optimization
- Database Management
- User Interface
- Physical Architecture





The purpose of **analysis** is to figure out what the business needs are.  
The purpose of **design** is to decide how to build the system.

هدف از تجزیه و تحلیل این است که بفهمیم نیازهای کسب و کار چیست. هدف از طراحی تصمیم گیری در مورد نحوه ساخت سیستم است.

# Layers

- Until this point in the development of our system, we have focused only on the problem domain; we have totally ignored the system environment (data management, user interface, and physical architecture).
- To successfully evolve the analysis model of the system into a design model of the system, we must add the system environment information.
- One useful way to do this, is to use *layers*.
- A *layer* represents an element of the software architecture of the evolving system.
- There should be a layer for each of the different elements of the system environment (e.g., data management, user interface, physical architecture).

توی دیزاین دوتا کار میکنیم یکی جزئیات رو به مدل تحلیل اضافه میکنیم و محیط رو داریم بهش اضافه میکنیم

وقتی که داریم محیط رو اضافه میکنیم ناگزیریم تقسیم بندی بکنیم

یکی از مشهورترین تقسیم بندی، تقسیم بندی لایه ای است

لایه ای: ینی لایه بندی میکنیم اون المان هایی که توی نرم افزارمون موثر است که هر لایه ای می تونه بیانگر مجموعه ابجکت های خاص اون لایه باشه



## Layers(Cnt'd)

- The idea of separating the different elements of the architecture into separate layers can be traced back to the MVC architecture of *Smalltalk*.
- Separate the application logic from the logic of the user interface. In this manner, it was possible to easily develop different user interfaces that worked with the same application.



# Software layers

- Foundation,
- Problem domain,
- Data management,
- Human–computer interaction,
- Physical architecture





# Foundation Layer

- Is a very uninteresting layer.
- It contains classes that are necessary for any object-oriented application to exist.
- They include classes that represent fundamental data types (e.g., integers, real numbers, characters, strings), classes that represent fundamental data structures, sometimes referred to as *container classes* (e.g., lists, trees, graphs, sets, stacks, queues), and classes that represent useful abstractions, sometimes referred to as *utility classes* (e.g., date, time, money).
- These classes are rarely, if ever, modified by a developer. They are simply used.
- Today, the classes found on this layer are typically included with the object-oriented development environments.

-  
این لایه زیرساخت همه لایه ها است

# Problem Domain Layer

- Is what we have focused our attention on up until now.
- At this stage in the development of our system, we need to further detail the classes so that we can implement them in an effective and efficient manner.

همون لایه ای است که مسئله ما اینجا تعریف میشه و تا الان روی همین کار می کردیم مثلا یوزکیس ها مرتبط با همین لایه بوده



# Data Management Layer

- Addresses the issues involving the persistence of the objects contained in the system.
- The types of classes that appear in this layer deal with how objects can be stored and retrieved.
- Classes contained in this layer are called the Data Access and Manipulation (DAM) classes.
- Allow the problem domain classes to be independent of the storage used and, hence, increase the portability of the evolving system.
- Some of the issues related to this layer include choice of the storage format and optimization.

لایه ای که اینجا داده میخواد سیو بشه و ذخیره بشه و بازیابی بشه ینی مدیریت داده می خواد انجام بشه  
کلاس های مرتبط با تعامل داده اینجاست و همه تعاملات با دیتابیس توی این لایه است

# Human–Computer Interaction Layer

- The primary purpose is to keep the specific user-interface implementation separate from the problem domain classes.
- This increases the portability of the evolving system.
- Typical classes found on this layer include classes that can be used to represent buttons, windows, text fields, scroll bars, check boxes, drop-down lists, and many other classes that represent user-interface elements.

-  
لایه ی فرانت سیستم است  
اینجا هم کلاس های زیادی وجود داره

# Physical Architecture Layer

- Addresses how the software will execute on specific computers and networks.
- This layer includes classes that deal with communication between the software and the computer's operating system and the network.

- حالا اون نرم افزار بیاد روی دیوایس های مختلف و شبکه چگونه اجرا بشه و منتقل بشه  
ینی کلاس های ارتباطی بین نرم افزار و سخت افزار و سیستم عامل و شبکه اینجا بررسی میشه

# Packages

- In UML, collaborations, partitions, and layers can be represented by a higher-level construct: a package.
- In fact, a package serves the same purpose as a folder on your computer.
- A *package* is a general construct that can be applied to any of the elements in UML models, group of use cases together to make the use-case diagrams easier to read and to keep the models at a reasonable level of complexity or a set of class and communication diagrams.

اینجا لایه بندی داریم نگا میکنیم  
این پکیج ها میتونن لایه ها باشن یا پارتیشن ها باشن  
پکیج ها مثل حالت های فولدرهای روی کامپیوتر هستن  
در حالت کلی حتی میتونن دسته از یوزکیس ها که با هم دیگه در تعامل هستن در قالب یک پکیج  
بیان بشن  
پکیج شامل یک مجموعه از المان های نرم افزاری است و هدف اصلیش هم غلبه بر پیچیدگی است



# Package Diagrams

- A diagram composed only of packages.
- In a package diagram, it is useful to depict *dependency relationship*.
- Dependency relationship represents the fact that a modification dependency exists between two packages. That is, it is possible that a change in one package could cause a change to be required in another package.

برای ترسیم از پکیج دیاگرام استفاده میکنیم

توی پکیج دیاگرام میایم پکیج های سیستم رو به تصویر می کشیم

یال های ما میشه وابستگی هایی که این پکیج ها با هم دیگه دارن

این وابستگی ها خیلی مهم هستن مخصوصا هرچی سیستم ما بزرگتر و حساس تر باشه این وابستگی

ها خیلی نقش مهمی دارن و یکی از جاها تغییرات است

# Elements of a package diagram

## A package:

- Is a logical grouping of UML elements.
- Is used to simplify UML diagrams by grouping related elements into a single higher-level element.



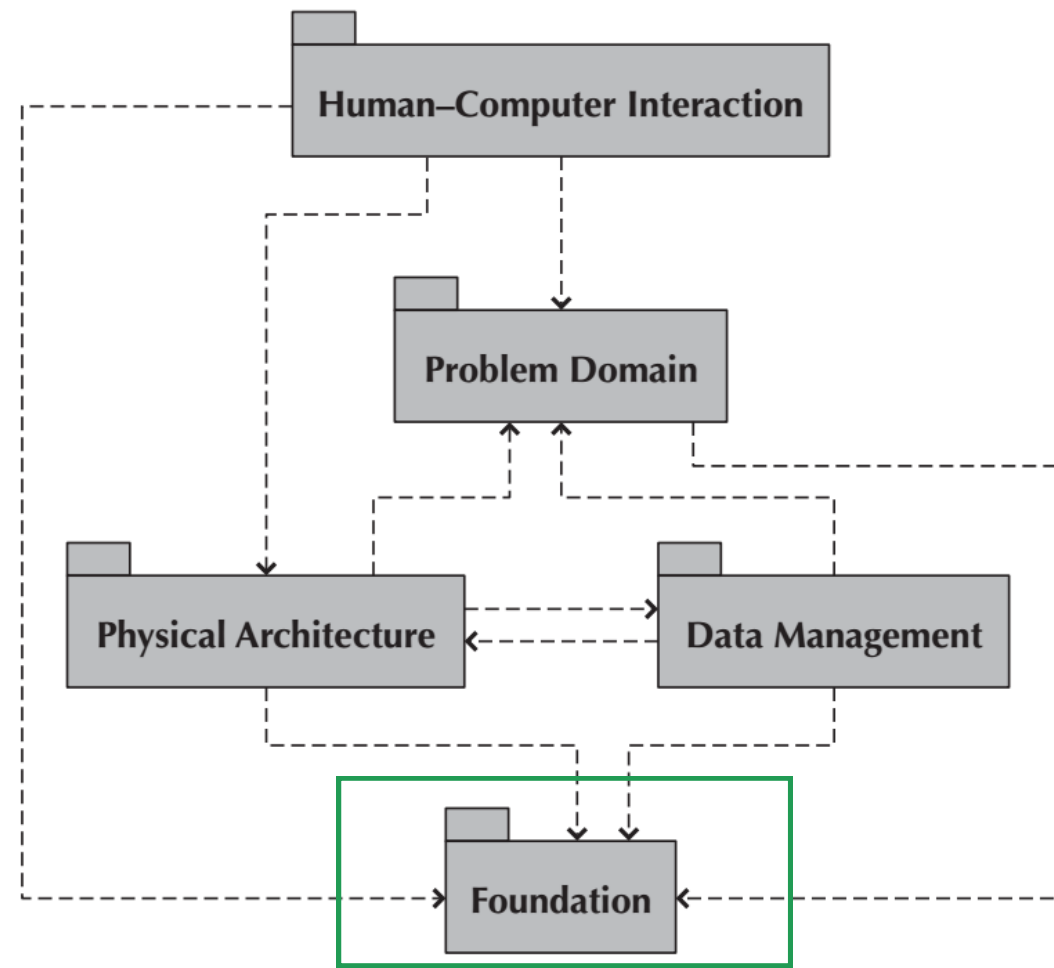
## A dependency relationship:

- Represents a dependency between packages: If a package is changed, the dependent package also could have to be modified.
- Has an arrow drawn from the dependent package toward the package on which it is dependent.



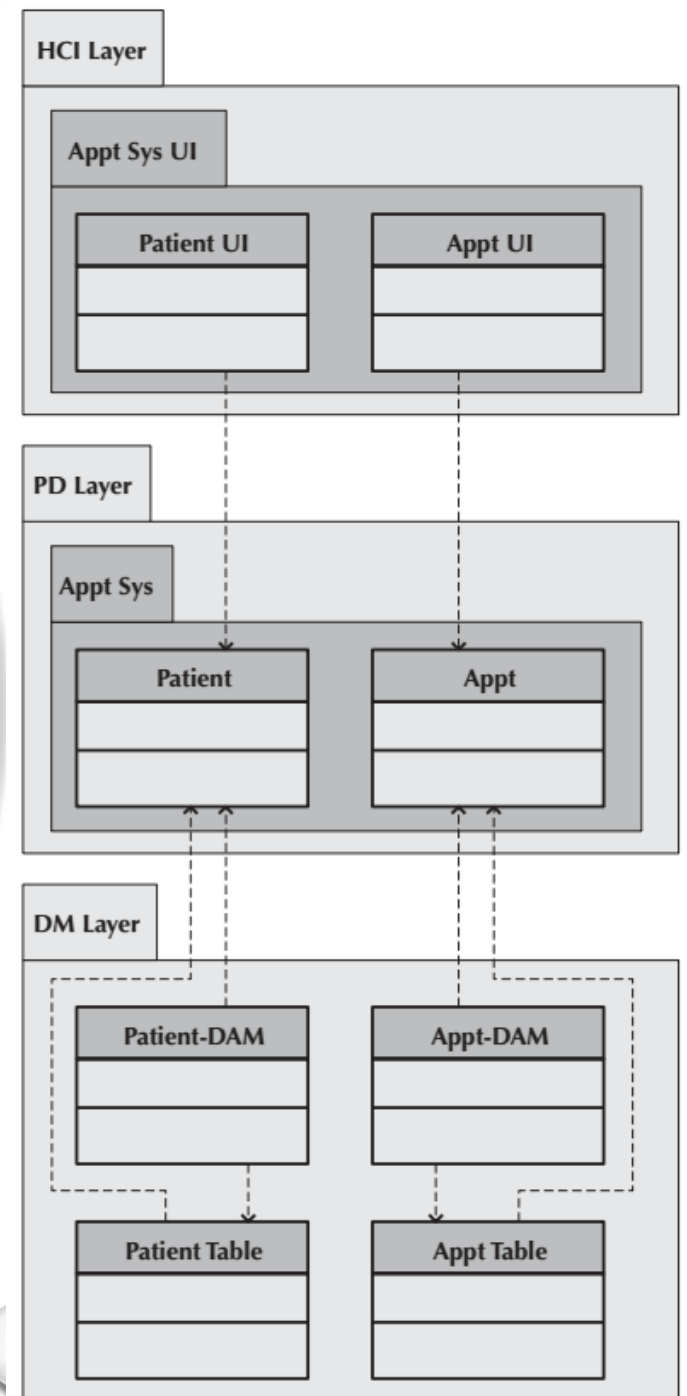
یاله: ارتباط وابستگی: وابستگی یک پکیج به پکیج دیگر--> اگر یک پکیج تغییر بکنه اون پکیج وابسته هم می تونه تغییر بکنه  
جهت یال از پکیج وابسته به اون پکیجی که بهش وابسته میشه

# Package Diagram of Dependency Relationships among Layers



- اینجا 5 تا لایه داریم و هر لایه می تونه در قالب یک پکیج بیان بشه

# An example of package diagram







# Verifying and validating package diagram

- Like all the previous models, package diagrams need to be verified and validated.
- First, the identified packages must make sense from a problem domain point of view.
- Second, all dependency relationships must be based on message-sending relationships on the communications diagram, and associations on the class diagram.

-

بعد از اینکه پکیج دیاگرام رو کشیدیم باید verifying , validation رو انجام بدیم ینی مطمئن بشیم از درستیش  
ینی مطمئن بشیم بیانگر مسئله اصلی ما است

# Design Strategies

- **Custom Development**
- **Packaged Software**
- **Outsourcing**





# Custom development or building a new system from scratch,

- Teams have complete control over the way the system looks and functions.
- Allows developers to be flexible and creative in the way they solve business problems.
- Is easier to change to include components that take advantage of current technologies that can support such strategic efforts.
- Building a system in-house also builds technical skills and functional knowledge within the company.
- As developers work with business users, their understanding of the business grows and they become better able to align IS with strategies and needs.
- These same developers climb the technology learning curve so that future projects applying similar technology require much less effort.

این استراتژی:

ینی از اول همه کار رو انجام میدیم و کنترلی روی روند تولید داریم و دولوپرها کاملاً نقش دارن و راحتتر می‌تونیم تغییرات رو مدیریت بکنیم و همینطوری اگه تکنولوژی‌های جدیدی باشه می‌تونه از مولفه‌های اونها استفاده کرده ضمن اینکه خود تیم هم رشد میکنه

# Custom development or building a new system from scratch(Cnt'd)

- Requires dedicated effort that involves long hours and hard work. Many companies have a development staff who already is overcommitted to filling huge backlogs of systems requests and just does not have time for another project. Also, a variety of skills—technical, interpersonal, functional, project management, and modeling—must be in place for the project to move ahead smoothly.
- The risks associated with building a system from the ground up can be quite high, and there is no guarantee that the project will succeed.
- Developers could be pulled away to work on other projects, technical obstacles could cause unexpected delays, and the business users could become impatient with a growing timeline.

زمان می خواد ینی ساعت های زیادی رو باید کار کرد  
ممکنه بعضی از تیم ها تایمی نداشته باشن روی این پروژه بذارن یا مهارت هایی میخواد که افراد  
هنوز ندارن  
ریسکش هم زیاده



# Packaged Software

- There are thousands of commercially available software programs that have already been written to serve a multitude of purposes.
- Similarly, most companies have needs that can be met quite well by packaged software, such as payroll or accounts receivable. It can be much more efficient to buy programs that have already been created, tested, and proven.
- Moreover, a packaged system can be bought and installed in a relatively short time when compared with a custom system. Plus, packaged systems incorporate the expertise and experience of the vendor who created the software.

استراتژی بعدی:

برای سیستم هایی که قبلا وجود داشته مثلا سیستم حقوقی میخوايم که اين سیستم قبلا نوشته شده و وجود داره منتها ممکنه کمی متفاوت باشه  
این برای ما می صرفه که بریم بخریم --> ریسک اینجا پایین تر است  
ولی انعطاف پذیر نیست

## Packaged Software(Cnt'd)

- Most packaged applications allow *customization*, or the manipulation of system parameters to change the way certain features work.
- *Systems integration* refers to the process of building new systems by combining packaged software, existing legacy systems, and new software written to integrate these.

-  
چالش بعدی:

# Outsourcing

- Hire an external vendor, developer, or service provider to create the system.
- This transfer requires two-way coordination, exchange of information, and trust.

-

استراتژی بعدی: برون سپاری  
ینی کرایه بکنیم یکسری ادما رو که برای ما سیستم بنویسن  
مدیریتش سخته

# Selecting a Design Strategy

	Use Custom Development When...	Use a Packaged System When...	Use Outsourcing When...
<b>Business Need</b>	The business need is unique.	The business need is common.	The business need is not core to the business.
<b>In-house Experience</b>	In-house functional and technical experience exists.	In-house functional experience exists.	In-house functional or technical experience does not exist.
<b>Project Skills</b>	There is a desire to build in-house skills.	The skills are not strategic.	The decision to outsource is a strategic decision.
<b>Project Management</b>	The project has a highly skilled project manager and a proven methodology.	The project has a project manager who can coordinate the vendor's efforts.	The project has a highly skilled project manager at the level of the organization that matches the scope of the outsourcing deal.
<b>Time frame</b>	The time frame is flexible.	The time frame is short.	The time frame is short or flexible.

در چه صورتی کدوم یکی از این استراتژی ها رو انتخاب میکنیم:

اول نوع نیازی که اون بازار داره --> اگر اون نیاز منحصر به فرد است بهتره که خودمون بنویسیم ولی اگر مشترکه پکیج رو می خریم

تجربه هایی که داریم توی تیم --> در صورتی اینو انتخاب میکنیم که تیم اون تجربه رو داشته باشه 1

توی 2 اون مهارت ها ما نیاز داریم چون شاید بخوایم تغییر بدیم ولی توی 3 لزومی نداره

مهارت های که نیاز داره برای اون پروژه خاص --> 1: تکنولوژی نو است و این خیلی کمک

میکنه و با اون پروژه رشد میکنیم 2: اون مهارت ها خیلی روتین هستن می تونیم از این استفاده

بکنیم 3: اگر مهارت ها اینجا مهارت های عمده ای باشن اگر این بی ارزه از این می رن وگرنه از

1 می رن

مدیریت overhead --> برای 1 خیلی زیاده 2: نه خیلی زیاد نیست 3: چرا اینجا هم overhead

داریم ینی بازم باید اینجا مدیریت بکنیم و بحث اعتماد است و این اطلاعات به چه صورتی باید بره

بیرون

زمان --> 1: اگر زمان وجود داشته باشه از این می ریم 2: اگر زمان کم باشه این می صرفه

3: اگر زمان انعطاف پذیر باشه و کم باشه از این می ریم



# Reference

- **Dennis, Wixon, Tegarden**, “System Analysis and Design, An Object Oriented Approach with UML”, 5th Edition, 2015.

