



Software Engineering I

Course Overview

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2021



Software in modern world

Software is
eating
the world!





NİLEM GENEL MÜDÜRLÜĞÜ [TR] | <https://kimlikdogrulama.gov.tr/captcha?returnUrl=%E7%9F%93%EA%9A%9F&signature=1.0%26id%3d10000000000000000000000000000000>

NVI.gov.tr



Software in modern world





Software in modern world





We can't run the modern world without software

- National infrastructures and utilities are controlled by computer-based systems.
- Most electrical products include a computer and controlling software.
- Industrial manufacturing and distribution is completely computerized.
- Entertainment, including computer games, and film and television, is software intensive.

زیرساختها و تاسیسات ملی توسط سیستم‌های کامپیوتری کنترل می‌شوند.

- بیشتر محصولات الکترونیکی شامل کامپیوتر و نرم افزار کنترل است
- تولید و توزیع صنعتی کاملاً کامپیوتری می‌باشد
- سرگرمی‌ها، از جمله بازی‌های رایانه‌ای، و فیلم و تلویزیون، نرم افزار فشرده است



Software Cost vs. Hardware Cost

- Business IT has changed significantly. Computing has become more distributed, portable, and personal.
- Even when hardware is issued by the company, employees use their own phones and computers to access email and apps.
- The business leverage has shifted to software, and budgets have followed.
- This change in spending is both a cause and effect of a broader shift of business IT from hardware to software and an important indicator of the future.

اهرم کسب و کار به سمت نرم افزار تغییر کرده است و بودجه نیز به دنبال آن بوده است. این تغییر در هزینه‌ها هم علت و هم نتیجه تغییر گسترده‌تر فناوری اطلاعات کسبوکار از سختافزار به نرمافزار و یک شاخص مهم برای آینده است.



Software costs

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

هزینه های نرم افزار اغلب بر هزینه های سیستم کامپیوترا غالب است. هزینه های نرم افزار در رایانه شخصی اغلب بیشتر از هزینه ساخت افزار است.

هزینه نگهداری نرم افزار بیشتر از توسعه آن است. برای سیستم هایی با عمر طولانی، هزینه های نگهداری ممکن است چندین برابر هزینه های توسعه باشد. مهندسی نرم افزار با توسعه نرم افزار مقرن به صرفه سروکار دارد.



References

- 1- Dennis, Wixon, Tegarden, “System Analysis and Design, An Object Oriented Approach with UML”, 5th Edition, 2015.
- 2- R. S. Pressman, B. R. Maxim, “Software Engineering, A Practitioner’s Approach”, 8th Edition, 2015.
- 3- Sommerville, I., “Software Engineering”, 10th Edition, 2015.
- 4- J. Sutherland, “Scrum handbook,” 2010.



Table of Contents

- ❖ Introduction to System
- ❖ Software Development Life Cycle
- ❖ Software development methodologies
- ❖ RUP and Agile
- ❖ Scrum
- ❖ Software Analysis
 - ❖ Functional modeling
 - ❖ Structural modelling
 - ❖ Behavioral modelling
- ❖ Software Design
- ❖ Design principles
- ❖ Database design
- ❖ User Interface design
- ❖ Architecture design
- ❖ Design patterns(maybe)



Grading Policy

- **$65 \pm 5\%$ on project.**
- **$20 \pm 5\%$ on Final exam.**
- **$15 \pm 5\%$ on Presentation.**
- **Late policy:** no credit for late work.



Course Overview

Course is actually three courses in one.

- Object-oriented approach
- Software analysis and design in the medium.
- Team working.



You will learn...

- How to **design software** using some powerful **abstraction mechanisms** and a collection of **patterns**;**·how to get it right, by construction and by modular reasoning;****·how to articulate your design ideas and critique other people's designs;**
- And on the way:
 - How to **think about a problem**.
 - How to **translate customer needs into diagrams**.
 - How to **analysis** the models and try to **improve** them.
 - How to work in a **team**.



What we expect from you

- Attend in the lab;
- Attend lectures;
- Present your proposal;
- Attend project reviews;
- Complete project activities;
- Help your team;



Course goals

- Think about the problem.
- Software Analysis.
- Design a software in an object-oriented manner.
- Design graphical user interfaces
- Work suitable in a team.



Life strategy

- Think in advance: **don't rush to code.**
- **Design** is more fun than **debugging!**
- **Focus** on ideas.
- **Don't be blinded by technology.**



For the next week

- Form a group with three or four members.
- Imagine your group as a company, select a name.
- Think about your project.



What we will talk about next...

- Introduction
- How to write a proposal.
- Introduction about System, Software Development Life Cycle(**SDLC**).



The background features a dark blue gradient with various 3D-style triangles of different sizes and orientations scattered across the surface. Some triangles are filled with a medium blue color, while others are outlined in light blue. In the top left corner, there is a small, thin-lined network graph consisting of a few blue circular nodes connected by thin grey lines.

THANKS!



Software Engineering I

Introduction

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2022



*No one could foresee that software would become **embedded in systems of all kinds:** transportation, medical, telecommunications, military, industrial, entertainment, . . .
the list is almost endless.*

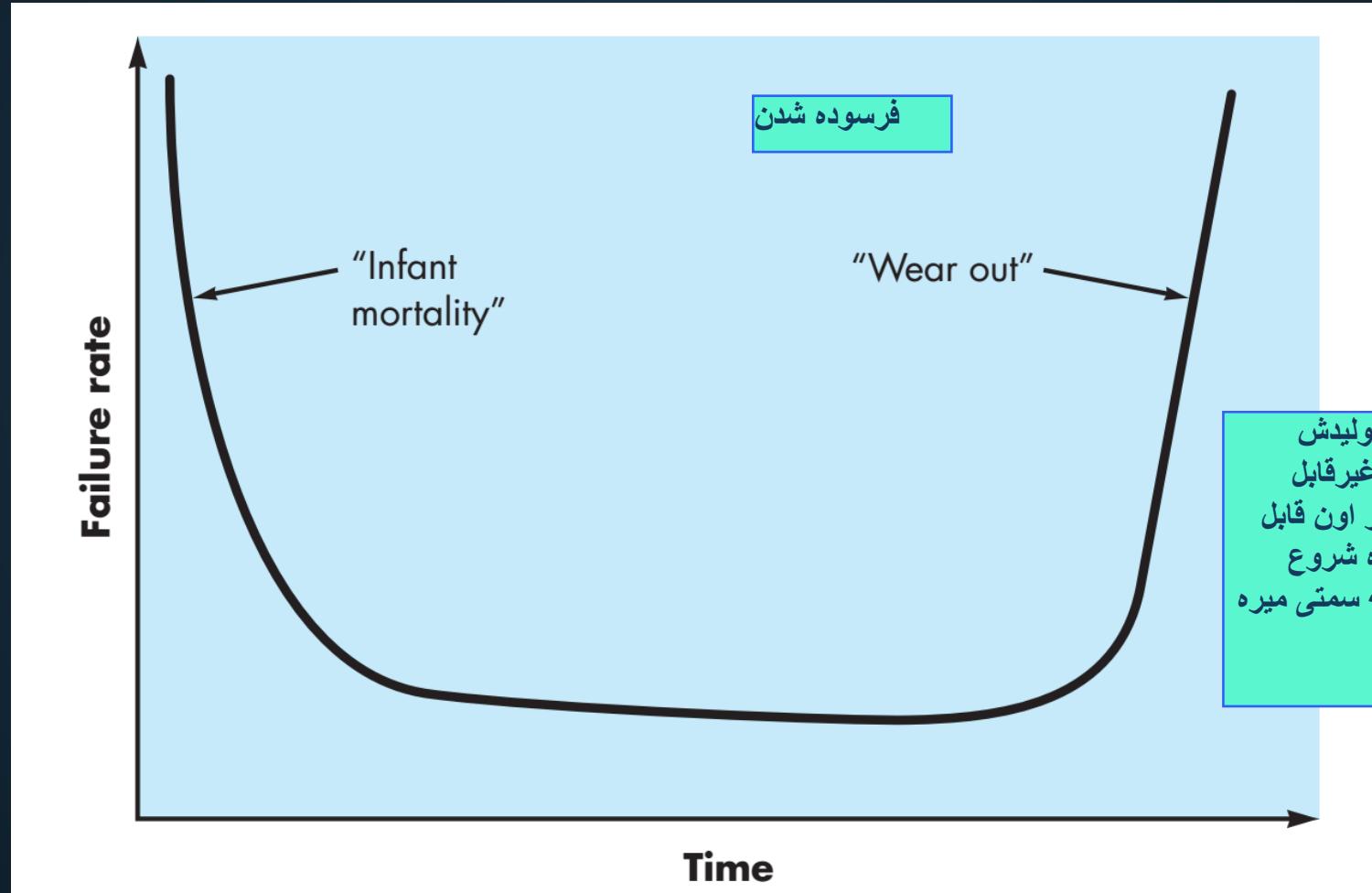


What is software?

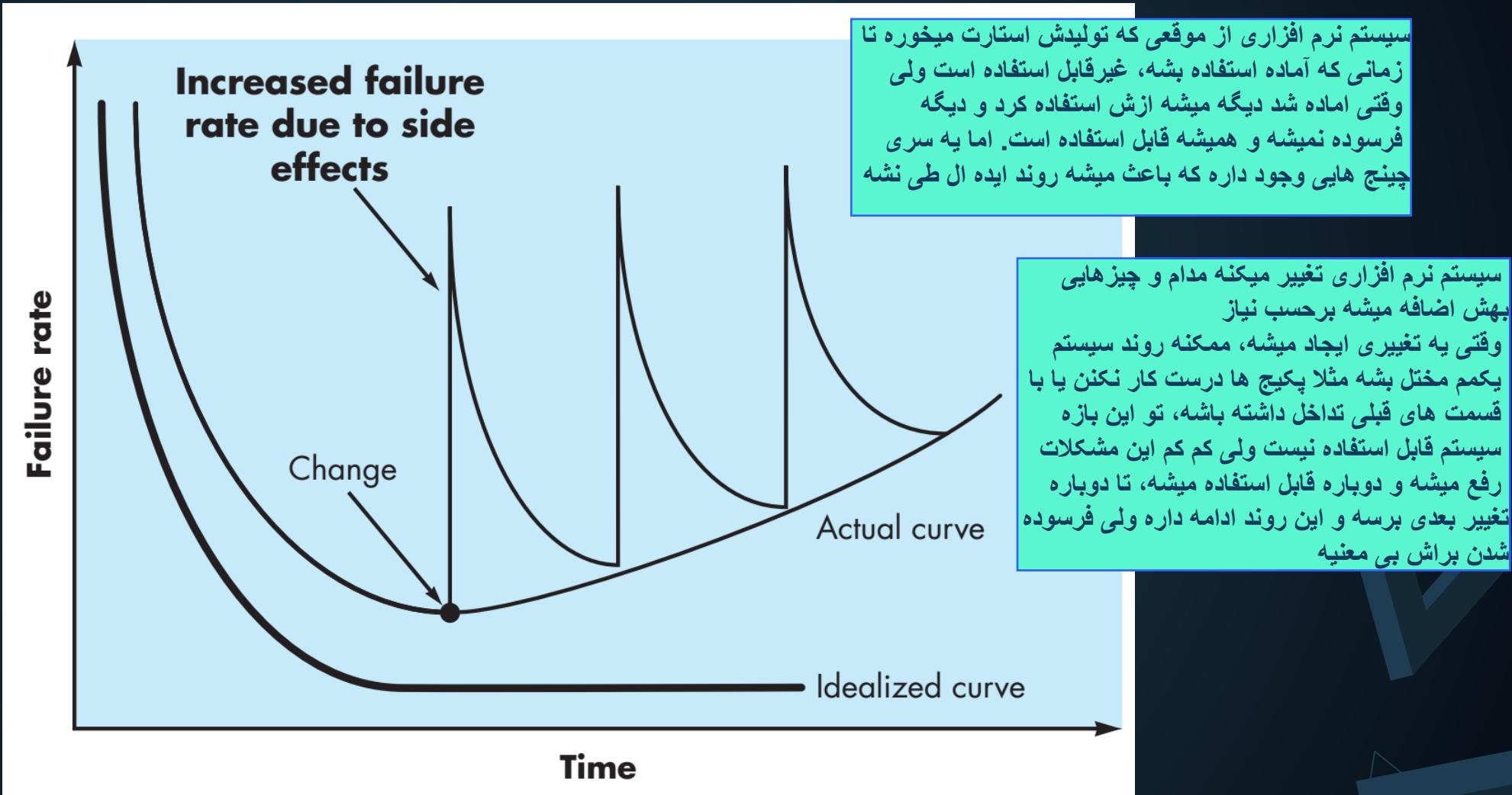
- Instructions (computer programs) that when executed provide desired features, function, and performance;
- Data structures that enable the programs to adequately manipulate information,
- Descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

Software is a logical rather than a physical system element.

Hardware failure rate as a function of time



Software failure rate as a function of time





Challenges

- Characteristics of software

- Intangible
- Changeable

ناملموس
تغییر پذیر

- Characteristics of software development

- Human-intensive
- Multi-disciplinary

انسان محور
چند رشته ای

سیستم ها باید با سرعت بیشتری ساخته و تحویل شوند.
سیستم های بزرگتر و حتی پیچیده تر مورد نیاز است.
سیستم ها باید قابلیت های جدیدی داشته باشند که قبلاً
غیرممکن تصور می شد.
روش های مهندسی نرم افزار موجود نمیتوانند با آن کنار
بیایند و تکنیک های مهندسی نرم افزار جدید باید برای
پاسخگویی به این خواسته های جدید توسعه یابند.

توقعت کم
نوشتن برنامه های کامپیوتری بدون استفاده از روش
ها و تکنیک های مهندسی نرم افزار نسبتاً آسان است.
بسیاری از شرکت ها از روش های مهندسی نرم
افزار استفاده نمی کنند. در نتیجه، نرم افزار آنها اغلب
گرانتر و کمتر از آنچه که باید باشد، قابل اعتماد است.



Software Failures – Main Reasons

- ***Increasing demands***

- Systems have to be built and delivered more **quickly**;
- Larger, even more **complex** systems are required;
- Systems have to have **new capabilities** that were previously thought to be impossible.
- Existing software engineering methods cannot cope and **new** software engineering techniques have to be developed to meet these new demands.

- ***Low expectations***

- It is relatively easy to write computer programs without using software engineering methods and techniques.
- Many companies do not use software engineering methods. Consequently, their software is often **more expensive** and less **reliable** than it should be.

We need better software engineering education and training to address this problem.



Software Engineering is NOT Programming!

Programming Vs. Engineering

Programming	Software Engineering
Personal activity (instrument)	Team activity (orchestra)
One aspect of software development	Large systems must be developed similar to other engineering practices
Concerned about accomplishing the objective of the program itself	Concerned about the entire solution, its feasibility, and future use



Professional Software Development

مشخصات برنامه، طراحی و تکامل

- Professional software, intended for use by someone apart from its developer, is usually **developed by teams rather than individuals**.
It is maintained and changed throughout its life.
- Software engineering is intended to **support professional software development**, rather than individual programming. It includes techniques that support program specification, design, and evolution.

- نرم افزارهای حرفه ای که برای استفاده توسط شخصی غیر از توسعه دهنده آن در نظر گرفته شده است، معمولاً توسط تیم ها به جای افراد توسعه می یابد.
- در طول عمر خود حفظ و تغییر می کند
- مهندسی نرم افزار برای پشتیبانی از توسعه نرم افزار حرفه ای به جای برنامه نویسی فردی در نظر گرفته شده است. این شامل تکنیک هایی است که از مشخصات، طراحی و تکامل برنامه پشتیبانی می کند



Professional Software Development(Cnt'd)

- Many people think that software is simply another word for computer programs. However, software is not just the programs themselves but also all associated documentation and configuration data that is required to make these programs **operate correctly**.
 - A professionally developed software system usually consists of **system documentation**, which describes the **structure** of the system; **user documentation**, which explains **how to use** the system.
- This is one of the important differences between professional and amateur software development.

یک سیستم نرم افزاری حرفه ای توسعه یافته معمولاً از مستندات سیستمی تشکیل شده است که ساختار سیستم را توصیف می کند. مستندات کاربر، که نحوه استفاده از سیستم را توضیح می دهد.

بسیاری از مردم فکر می کنند که نرم افزار به سادگی کلمه دیگری برای برنامه های کامپیوتری است. با این حال، نرمافزار فقط خود برنامهها نیستند، بلکه تمام اسناد و داده های پیکربندی مرتبط نیز هستند که برای عملکرد صحیح این برنامهها مورد نیاز هستند.



Software Products

- Software engineers are concerned with developing software products.
- Kinds of software products
 - *Generic products: The specification of what the software should do is owned by the software developer and decisions on software change are made by the developer.*
 - *Customized products: The specification of what the software should do is owned by the customer for the software and they make decisions on software changes that are required.*

محصولات عمومی: مشخصات کاری که نرم افزار باید انجام دهد متعلق به توسعه دهنده ◦
نرم افزار است و تصمیمات در مورد تغییر نرم افزار توسط توسعه دهنده اتخاذ می شود

محصولات سفارشی شده: مشخصات کاری که نرم افزار باید انجام دهد متعلق به مشتری ◦
برای نرم افزار است و آنها در مورد تغییرات نرم افزاری مورد نیاز تصمیم می گیرند



Software Engineering(I)

مهندسی نرم افزار یک رشته مهندسی است که با تمام جنبه های تولید نرم افزار از مرحله مشخصات سیستم تا حفظ و نگهداری سیستم پس از استفاده از آن سروکار دارد.

- Software engineering is an **engineering discipline** that is concerned with **all aspects of software production** from the **early stages** of **system specification** through to maintaining the system after it has gone into use.
 - *Engineering discipline* - engineers make things work. They apply **theories, methods, and tools** where these are appropriate. However, they use them **selectively** and always try to **discover solutions** to problems even when there are no applicable theories and methods. Engineers also recognize that they must work to **organizational and financial constraints** so they look for solutions within these **constraints**.
 - *All aspects of software production* - software engineering is not just concerned with the technical processes of software development. It also includes activities such as **software project management** and the **development** of tools, methods, and theories to support software production.

همچنین شامل فعالیت هایی مانند مدیریت پژوهه نرم افزاری و توسعه ابزارها، روش ها و تئوری ها برای پشتیبانی از تولید نرم افزار



Software Engineering(II)

- Doing the **right thing**
 - ❖ Software that users **want and need**
 - ❖ Software that **benefits** society
- Doing the **thing right**
 - ❖ Following a **good software process**
 - ❖ Developing your **programming skills**

کار درست را انجام دادن •
نرم افزاری که کاربران می خواهند و نیاز دارند
نرم افزاری که به نفع جامعه است
انجام درست کار •
دنبال کردن یک فرآیند نرم افزاری خوب
مهارت های برنامه نویسی خود را توسعه دهید



Software Engineering(III)

IEEE Computer Society Definition:

“Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software.”

مهندسی نرم افزار کاربرد یک رویکرد سیستماتیک، منظم و قابل سنجش برای توسعه، بهره برداری و نگهداری نرم افزار و مطالعه این رویکردها است. یعنی کاربرد مهندسی در نرم افزار



Engineering Discipline

به طور کلی، مهندسان نرم افزار یک رویکرد سیستماتیک و سازمان یافته برای کار خود اتخاذ می کنند، زیرا این اغلب موثرترین راه برای تولید نرم افزار با کیفیت بالا است.

مهندسی در مورد بدست آوردن نتایج قابل تکرار با کیفیت مورد نیاز در برنامه و بودجه است.

Engineering is about getting repeatable results of the required quality within the schedule and budget.

This often involves making compromises—engineers cannot be perfectionists.

People writing programs for themselves, however, can spend as much time as they wish on program development.

In general, software engineers adopt a systematic and organized approach to their work, as this is often the most effective way to produce high-quality software.

However, engineering is all about selecting the most appropriate method for a set of circumstances so a more creative, less formal approach to development may be effective in some circumstances.



General Issues of software

ناهمگونی به طور فزاینده ای، سیستم ها باید به عنوان سیستم های توزیع شده در سراسر شبکه ها که شامل انواع مختلف رایانه و دستگاه های تلفن همراه هستند، کار کنند

- *Heterogeneity* increasingly, systems are required to operate as distributed systems across networks that include **different types** of computer and mobile devices. As well as running on **general-purpose computers**, software may also have to execute on mobile phones.
- You often have to **integrate** new software with **older legacy systems** written in different programming languages.
- The challenge here is to develop techniques for building **dependable software** that is **flexible** enough to cope with this heterogeneity.

چالش در اینجا توسعه تکنیک هایی برای ساختن نرم افزار قابل اعتماد است که به اندازه کافی انعطاف پذیر باشد تا بتواند با این ناهمگونی کار بیاید

شما اغلب مجبور هستید نرم افزار جدید را با سیستم های قدیمی که به زبان های برنامه نویسی مختلف نوشته شده اند، ادغام کنید



آنها باید به گونه ای تکامل یابند که زمان
موردنیاز نرم افزار برای ارائه ارزش به
مشتریان کاهش یابد

General Issues of software(Cnt'd)

با توسعه اقتصادهای نوظهور و در دسترس قرار گرفتن فناوری های جدید، تجارت و
جامعه به سرعت در حال تغییر هستند.

- *Business and social change* business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.
 - Many traditional software engineering techniques are time consuming and delivery of new systems often takes longer than planned. They need to evolve so that the time required for software to deliver value to its customers is reduced.
- *Security and trust* as software is intertwined with all aspects of our lives, it is essential that we can trust that software. This is especially true for remote software systems accessed through a web page or web service interface.
 - We have to make sure that malicious users cannot attack our software and that information security is maintained.

از آنجایی که نرم افزار با تمام جنبه های زندگی ما در
هم تنیده است، ضروری است که بتوانیم به آن نرم افزار
اعتماد کنیم.

ما باید مطمئن شویم که کاربران مخرب نمی توانند به نرم افزار
ما حمله کنند و امنیت اطلاعات حفظ می شود



Software Engineering Diversity

نحوه اجرای این رویکرد سیستماتیک بسته به سازمانی که نرم افزار را توسعه می دهد، نوع نرم افزار و افراد درگیر در فرآیند توسعه به طور چشمگیری متفاوت است.

- Software engineering is a systematic approach to the production of software that takes into account practical cost, schedule, and dependability issues, as well as the needs of software customers and producers.
 - How this systematic approach is actually implemented varies dramatically depending on the organization developing the software, the type of software, and the people involved in the development process.
 - There are no universal software engineering methods and techniques that are suitable for all systems and all companies. Rather, a diverse set of software engineering methods and tools has evolved over the past 50 years.

مهندسی نرم افزار یک رویکرد سیستماتیک برای تولید نرم افزار است که هزینه عملی، زمان بندی و مسائل مربوط به قابلیت اطمینان و همچنین نیازهای مشتریان و تولیدکنندگان نرم افزار را در نظر می بگیرد.



Software Engineering Diversity(Cnt'd)

- You use different software engineering techniques for each type of system because the software has quite different characteristics.
 - For example, an embedded control system in an automobile is **safety-critical** and is burned into rom when installed in the vehicle. It is therefore **very expensive to change**. Such a system needs **very extensive verification and validation** so that the chances of having to recall cars after sale to fix software problems are minimized. **User interaction is minimal** (or perhaps nonexistent) so there is no need to use a development process that relies on user interface prototyping.
 - For a web-based system, an approach based on **iterative development and delivery** may be appropriate, with the system being composed of **reusable components**.



Software Engineering Diversity(Cnt'd)

سازمان توسعه‌دهنده نرم‌افزار باید فرآیند توسعه را برنامه‌ریزی کند و ایده‌های روشی از تولید و زمان تکمیل آن داشته باشد

قابلیت اطمینان و عملکرد برای همه انواع سیستم‌ها مهم است

- There are software engineering fundamentals that apply to all types of software systems.
 - They should be developed using a managed and understood development process. The organization developing the software should plan the development process and have clear ideas of what will be produced and when it will be completed. Of course, different processes are used for different types of software.
 - Dependability and performance are important for all types of systems. Software should be safe in its operation and, as far as possible, should be secure against external attack. The system should perform efficiently and should not waste resources.

نرم افزار باید همانطور که انتظار می‌رود، بدون خرابی رفتار کند و در صورت نیاز برای استفاده در دسترس باشد.

باید در عملکرد خود این باشد
باید در برابر حمله خارجی این باشد
سیستم باید کارآمد عمل کند و منابع را هدر ندهد.



Software Engineering Diversity(Cnt'd)

درک و مدیریت مشخصات و الزامات نرم افزار (آنچه نرم افزار باید
انجام دهد) مهم است

- Understanding and managing the software specification and requirements (what the software should do) are important.
 - You have to know what different customers and users of the system expect from it and you have to manage their expectations so that a useful system can be delivered within budget and to schedule.
- You should make as effective use as possible of existing resources. This means that, where appropriate, you should reuse software that has already been developed rather than write new software.

شما باید تا حد امکان از منابع موجود استفاده موثر داشته باشید



Essential attributes of good software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

بنابراین کارایی شامل پاسخگویی، زمان پردازش، استفاده از حافظه و غیره است.

Safety and reliability

ایمنی و قابلیت اطمینان

قابلیت اطمینان مربوط به انطباق با مشخصات و ارائه خدمات است.
احتمال عملکرد سیستم بدون خرابی در یک زمان مشخص در یک محیط معین برای یک هدف معین

- ✧ Safety and reliability are related but distinct
 - In general, reliability is necessary but not sufficient conditions for system safety.
- ✧ Reliability is concerned with conformance to a given specification and delivery of service.
 - The probability of failure-free system operation over a specified time in a given environment for a given purpose
- ✧ Safety is concerned with ensuring system cannot cause damage irrespective of whether or not it conforms to its specification.
 - System reliability is essential for safety but is not enough

ایمنی به این موضوع مربوط می شود که اطمینان حاصل شود که سیستم صرف نظر از انطباق یا عدم انطباق با مشخصات آن نمی تواند آسیب وارد کند.



General principles

یک سیستم نرم افزاری به یک دلیل وجود دارد: ارائه ارزش به کاربرانش
یک چشم انداز روشن برای موفقیت یک پروژه نرم افزاری ضروری است
کدنویسی با نگرانی برای کسانی که باید سیستم را حفظ و گسترش دهند.

1. *The Reason It All Exists*
 - A software system exists for one reason: *to provide value to its users.*
2. *Keep It Simple, Stupid!*
 - All design should be *as simple as possible, but no simpler.*
3. *Maintain the Vision*
 - A *clear vision* is essential to the success of a software project.
4. *What You Produce, Others Will Consume*
 - Specify with an eye to the users. Design, keeping the implementers in mind. Code with concern for those that must *maintain* and *extend* the system.



General principles(Cnt'd)

هزینه را کاهش می دهد و ارزش اجزای قابل استفاده مجدد و سیستم هایی که در آنها گنجانده شده اند را افزایش می دهد.

5. Be Open to the Future

- A system with a **long lifetime** has **more value**.
- These systems must be **ready** to **adapt** to the **changes**.

6. Plan Ahead for Reuse

- It **reduces the cost** and **increases the value** of both the reusable components and the systems into which they are incorporated.

7. Think!

- Placing **clear, complete thought** before action almost always produces better results.



References

- **Dennis, Wixon, Tegarden, “System Analysis and Design, An Object Oriented Approach with UML”, 5th Edition, 2015.**
- **<https://www.vgarousi.com>, accessed on 28th September, 2020.**
- Sommerville, I., “Software Engineering”, 10th Edition, 2015.



What we will talk about next...

- How to write a proposal.
- Introduction about System, **Software Development Life Cycle(SDLC)**.



Software Engineering I

System Development Life Cycle(SDLC)

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

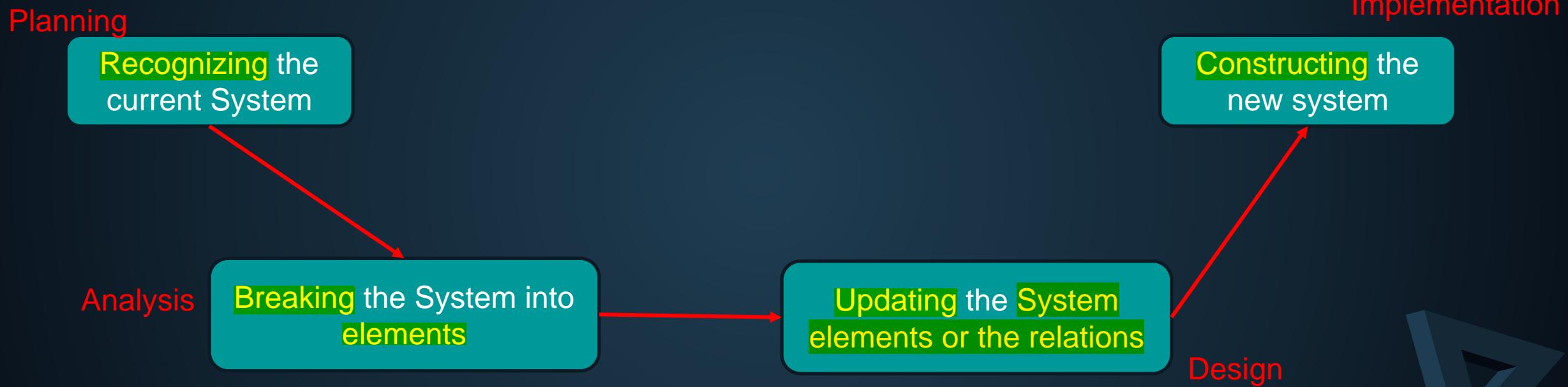
2022

System and System engineering

- A **system** is a collection of elements related in a way that allows a common objective to be accomplished.
 - In computer systems, these **elements** include **hardware**, **software**, **people**, **facilities**, and **processes**.
- **System engineering** is the **practical application** of scientific, engineering, and management **skills** necessary to transform an **operational need** into a description of a **system configuration** that best satisfies that need.
 - It is a generic **problem-solving process** that applies to the overall technical management of a system development project.

مهندسی سیستم عبارت است از کاربرد عملی مهارت های علمی، مهندسی و مدیریتی لازم برای تبدیل یک نیاز عملیاتی به توصیفی از پیکربندی سیستم که بهترین نیاز را پرآورده می کند.

Fundamental phases



Important points

یک فرآیند پالایش تدریجی است

- Is a process of *gradual refinement*.
 - The deliverables produced in the **analysis** phase provide a **general idea** of the shape of **the new system**. These deliverables are used as **input** to the **design** phase, which then **refines** them to produce a set of deliverables that describes in much **more detailed terms** exactly how the system will be built.
 - Each phase **refines** and **elaborates** on the work done previously.

هر مرحله کارهای انجام شده قبلی را اصلاح و شرح می دهد

1- Planning

- Is the fundamental process of understanding *why* an information system should be built and determining *how* the project **team** will go about building it.
- It has **two steps**.
 1. project **initiation**
 2. project **management**

فرآیند اساسی درک اینکه چرا یک سیستم اطلاعاتی باید ساخته شود و تعیین اینکه چگونه تیم پروژه برای ساخت آن اقدام خواهد کرد، است

دو مرحله دارد

شروع پروژه 1.

مدیریت پروژه 2.

1-1- Project initiation

:ارزش تجاری سیستم برای سازمان مشخص می شود
چگونه هزینه ها را کاهش می دهد یا درآمد را افزایش می دهد؟

یک درخواست سیستم خلاصهای از یک نیاز تجاری را ارائه میکند و توضیح میدهد که چگونه سیستمی که از نیاز پشتیبانی میکند ارزش تجاری ایجاد میکند.

- During *project initiation*, the **system's business value** to the organization is identified: How will it **lower costs** or **increase revenues**? Most **ideas for new systems** come from outside the **IS area** (e.g., from the marketing department, accounting department) in the form of a **system request**.
 - A system request presents a brief summary of a **business need**, and it explains **how** a system that supports the need will create **business value**.
- The **IS department** works together with the person or department that **generated** the **request** (called the *project sponsor*) to conduct a **feasibility analysis**.
- The **system request** and **feasibility analysis** are presented to an information systems **approval committee** to decide whether the project should be undertaken.

درخواست سیستم و تحلیل امکان سنجی به کمیته تایید سیستم های اطلاعاتی ارائه می شود تا تصمیم بگیرد که آیا پروژه باید انجام شود یا خیر.

1-2- Project management

- Once the project is **approved**, it enters *project management*.
- During project management, the **project manager** creates a **workplan**, staffs the project, and puts **techniques** in place to help the project team **control** and **direct** the project through the entire SDLC.
- The deliverable for project management is a **project plan**, which describes how the project team will go about **developing** the system.

قابل تحویل برای مدیریت پروژه یک طرح پروژه است که نحوه توسعه سیستم را توسط تیم پروژه توضیح می دهد.

در طول مدیریت پروژه، مدیر پروژه یک برنامه کاری ایجاد می کند، پروژه را مدیریت می کند و تکنیک هایی را برای کمک به در نظر **SDLC** تیم پروژه در کنترل و هدایت پروژه از طریق کل می گیرد.

2- Analysis

- Answers the questions of **who** will **use** the system, **what** the system will do, and **where** and **when** it will be used.
- During this phase, the project team investigates any **current system(s)**, identifies opportunities for **improvement**, and develops a **concept** for the new system.
- It has **three steps**.
 1. Selecting **analysis strategy**
 2. Gathering **requirements**
 3. Preparing **advanced proposal**

در طول این مرحله، تیم پروژه هر سیستم (های) فعلی را بررسی میکند. فرصت‌های بهبود را شناسایی میکند و مفهومی برای سیستم جدید ایجاد میکند. سه مرحله دارد.
انتخاب استراتژی تجزیه و تحلیل . 1. جمع آوری الزامات . 2. تهیه پروپوزال پیشرفته . 3.

2-1- Analysis strategy

- An *analysis strategy* is developed to guide the project team's **efforts**.
- Such a strategy usually includes an **analysis** of the current system (called the *as-is system*) and its **problems** and then ways to design a new system (called the *to-be system*).
- Types of strategies
 - Data-oriented
 - Process-oriented
 - Object-oriented

یک استراتژی تجزیه و تحلیل برای هدایت تلاش های تیم پروژه ایجاد می شود. چنین استراتژی معمولاً شامل تجزیه و تحلیل سیستم فعلی و مشکلات آن و سپس راه های طراحی یک سیستم جدید است.

2-2- Requirements gathering

تجزیه و تحلیل این اطلاعات - در ارتباط با ورودی از طرف حامی پروژه و بسیاری از افراد دیگر - منجر به توسعه مفهومی برای یک سیستم جدید می شود

- Through **interviews** or **questionnaires**.
- The analysis of this information—in conjunction with **input** from the project **sponsor** and many other people—leads to the development of a concept for a new system.
- The **system concept** is then used as a **basis** to develop a set of **business analysis models**, which describe **how the business will operate** if the new system is developed.

سپس مفهوم سیستم به عنوان مبنایی برای توسعه مجموعه‌های از مدل‌های تحلیل کسبوکار استفاده می‌شود، که توضیح میدهد در صورت توسعه سیستم جدید، چگونه کسبوکار عمل خواهد کرد.

2-3- Advanced system proposal

- The **analyses**, **system concept**, and **models** are combined into a document called the advanced system proposal, which is presented to the **project sponsor** and other **key decision makers** who decide whether the project should continue to move forward.

تجزیه و تحلیل ها، مفهوم سیستم و مدل ها در یک سند
ترکیب می شوند

3- Design

- Decides *how* the system will operate, in terms of the hardware, software, and network infrastructure; the user interface, forms, and reports; and the specific programs, databases, and files that will be needed.
- The design phase has five steps.
 - Selecting design strategy
 - Designing the physical architecture of the system
 - Designing interface of the system
 - Designing database and file specifications
 - Designing the program design

تصمیم می گیرد که سیستم از نظر ساخت افزار، نرم افزار و زیرساخت شبکه چگونه کار کند. رابط کاربری، فرم ها و گزارش ها؛ و برنامه ها، پایگاه داده ها و فایل های خاصی که مورد نیاز خواهد بود.

Steps of Design(I)

سیستم توسط برنامه نویسان خود شرکت توسعه خواهد یافت
سیستم به یک شرکت دیگر (معمولًاً یک شرکت مشاوره) برون سپاری ○
خواهد شد.
شرکت بسته نرم افزاری موجود را خریداری خواهد کرد ○

- *Design strategy*
 - System will be developed by the company's **own programmers**,
 - System will be **outsourced** to another **firm** (usually a consulting firm),
 - Company will **buy an existing software package**.
- *Physical architecture design* describes the **hardware**, **software**, and **network infrastructure** to be used.
- The *interface design* specifies how the **users** will **move through** the **system** (e.g., **navigation methods** such as menus and on-screen buttons) and the **forms and reports** that the system will use.

Steps of Design(II)

- The *database and file specifications* are developed. These define exactly what data will be stored and where they will be stored.
- *Program design*, which defines the programs that need to be written and exactly what each program will do.

End of Design

- This **collection** of **deliverables** (architecture design, interface design, database and file specifications, and program design) is the ***system specification*** that is handed to the **programming team** for implementation. مشخصات سیستم
- At the end of the design phase, the **feasibility analysis** and **project plan** are **reexamined** and **revised**, and another decision is made by the project **sponsor** and **approval committee** about whether to terminate the project or continue.

4- Implementation

- The system is actually built (or purchased).
- This phase has three steps.
 1. System **construction**.
 2. **Installation**.
 3. preparing a **support plan** for the system.

ساخت سیستم.

نصب و راه اندازی.

تهیه طرح پشتیبانی از سیستم.

Phases of Implementation

- **System construction** is the first step. The system is **built** and **tested** to ensure that it performs as designed. Because the cost of bugs can be immense, **testing** is one of the **most critical** steps in implementation.
- **Installation** is the process by which the **old system is turned off** and the new one is turned on. One of the most important aspects of **conversion** is the development of a **training plan** to **teach users** how to use the new system and help manage the changes caused by the new system.
- The analyst team establishes a **support plan** for the system. This plan usually includes a **formal** or **informal** post-implementation **review** as well as a systematic way for identifying major and minor **changes** needed for the system.

این طرح معمولاً شامل بررسی رسمی یا غیررسمی پس از
اجرا و همچنین روشی سیستماتیک برای شناسایی تغییرات
عمده و جزئی مورد نیاز برای سیستم است.



References

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



What we will talk about next...

- Process models



Software Engineering I

Process Models

Dr. Elham Mahmoudzadeh
Isfahan University of Technology
mahmoudzadeh@iut.ac.ir
2022



Software Process models

- Describes the sequences of SDLC steps.
- Is a sequence of activities that leads to the production of a software product.

دنباله ای از فعالیت هایی است که منجر به تولید یک محصول نرم افزاری می شود



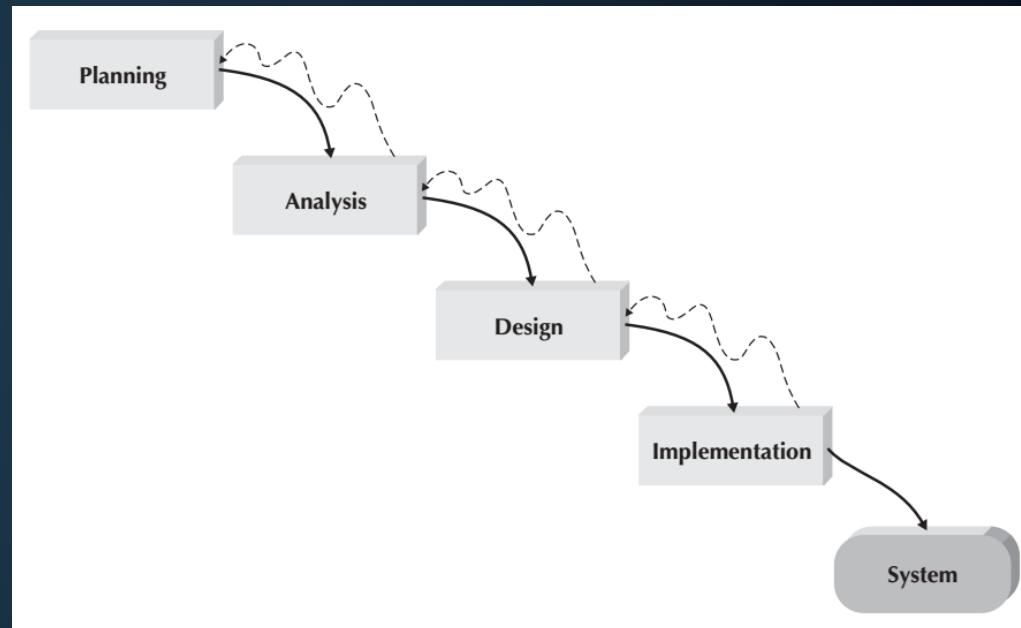
Process Models types

- Structured
 - Waterfall
 - parallel
- RAD
 - Phased
 - Prototyping
 - Throwaway-prototyping
- Agile



Waterfall(I)

- Proceed in sequence from one phase to the next.
- The key deliverables for each phase are typically very long (often hundreds of pages in length) and are presented to the project sponsor for approval as the project moves from phase to phase. Once the sponsor approves the work that was conducted for a phase, the phase ends and the next one begins.
- It moves forward from phase to phase in the same manner as a waterfall. Although it is possible to go backward in the SDLC (e.g., from design back to analysis), it is extremely difficult.
- Because of the cascade from one phase to another, this model is known as the ‘waterfall model.’





طراحی باید قبل از شروع برنامه نویسی به طور کامل مشخص شود.

زمان زیادی بین تکمیل پیشنهاد سیستم در مرحله تجزیه و تحلیل و تحويل سیستم (معمولًا چندین ماه یا سال) می گذرد.

Waterfall(II)

نیازهای سیستم را مدت‌ها قبل از شروع برنامه نویسی شناسایی می کند. این تغییرات در الزامات را در طول پروژه به حداقل می رساند.

- Key advantages

- It identifies system requirements long before programming begins.
- It minimizes changes to the requirements as the project proceeds.

- Key disadvantages

- Design must be completely specified before programming begins
- A long time elapses between the completion of the system proposal in the analysis phase and the delivery of the system (usually many months or years).
- If the project team misses important requirements, expensive post-implementation programming may be needed. A system can also require significant rework because the business environment has changed from the time when the analysis phase occurred.

اگر تیم پروژه الزامات مهم را از دست بدهد، ممکن است به برنامه‌ریزی گرانقیمت پس از پیدا‌هسازی نیاز باشد. یک سیستم همچنین میتواند نیاز به تجدید نظر قبل توجهی داشته باشد، زیرا محیط کسبوکار از زمانی که مرحله تجزیه و تحلیل رخ داده تغییر کرده است.



Waterfall Main drawback

اشکال اصلی مدل آبشار دشواری انطباق با تغییرات پس از انجام فرآیند است.

در اصل، یک مرحله باید قبل از رفتن به مرحله بعدی کامل شود.

- The **main drawback** of the waterfall model is the **difficulty of accommodating change after the process is underway**.
- In principle, a phase has to be complete before moving onto the next phase.



Waterfall Model - Usage

در اصل، مدل آبشار تنها زمانی باید مورد استفاده قرار گیرد که الزامات به خوبی درک شده باشند و بعيد است که در طول توسعه سیستم به طور اساسی تغییر کند.

- In principle, the waterfall model should **only be used** when the **requirements are well understood** and **unlikely to change** radically during system development.
- The waterfall model is mostly used for **large systems** engineering projects where a system is developed at **several sites**.
- In those circumstances, the **plan-driven** nature of the waterfall model helps **coordinate** the work.

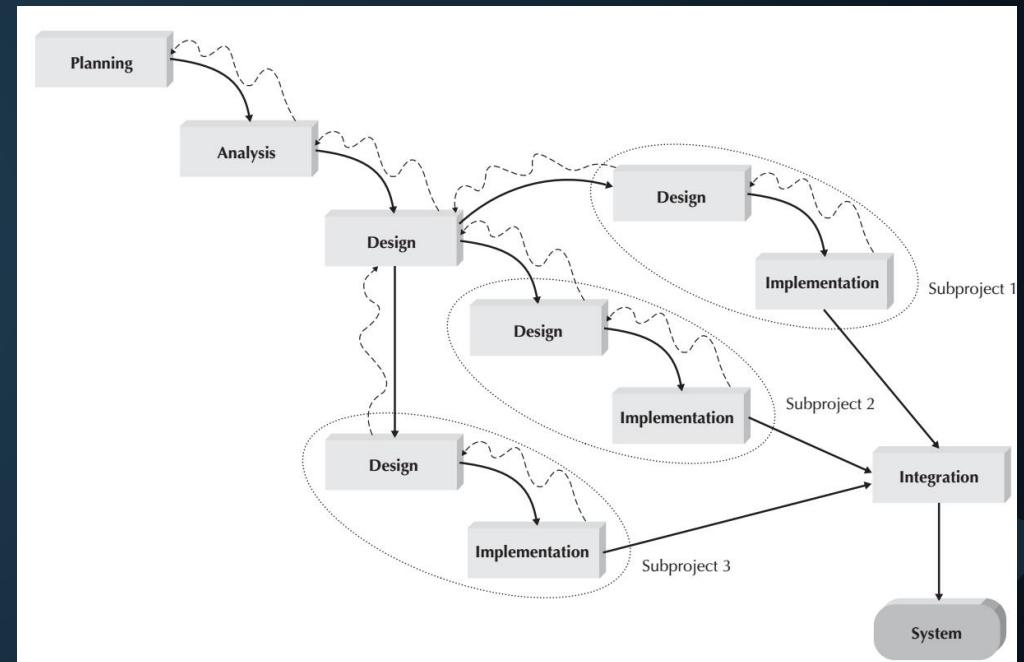
در آن شرایط، ماهیت طرح محور مدل آبشار به هماهنگی کار کمک می کند.



Parallel Development(I)

تلاش برای رسیدگی به مشکل تأخیرهای طولانی بین مرحله تجزیه و تحلیل و تحویل سیستم.

- Attempts to address the problem of **long delays** between the analysis phase and the delivery of the system.
- Instead of doing design and implementation in sequence, it performs **a general design** for the **whole system**.
- Then **divides** the project into a series of **distinct subprojects** that can be designed and implemented in **parallel**.
- Once all **subprojects** are **complete**, the separate pieces are **integrated** and the system is delivered.





Parallel Development(III)

- The primary advantage is that it can reduce the time to deliver a system;
- However, sometimes the subprojects are not completely independent; design decisions made in one subproject can affect another.
- At the end of the project, it requires significant integration efforts.

مزیت اصلی این است که می تواند زمان تحویل یک سیستم را کاهش دهد.
با این حال، گاهی اوقات پروژه های فرعی کاملاً مستقل نیستند.
تصمیمات طراحی گرفته شده در یک پروژه فرعی می تواند دیگری را تحت تأثیر قرار دهد.
در پایان پروژه، نیاز به تلاش های ادغام قابل توجهی دارد.



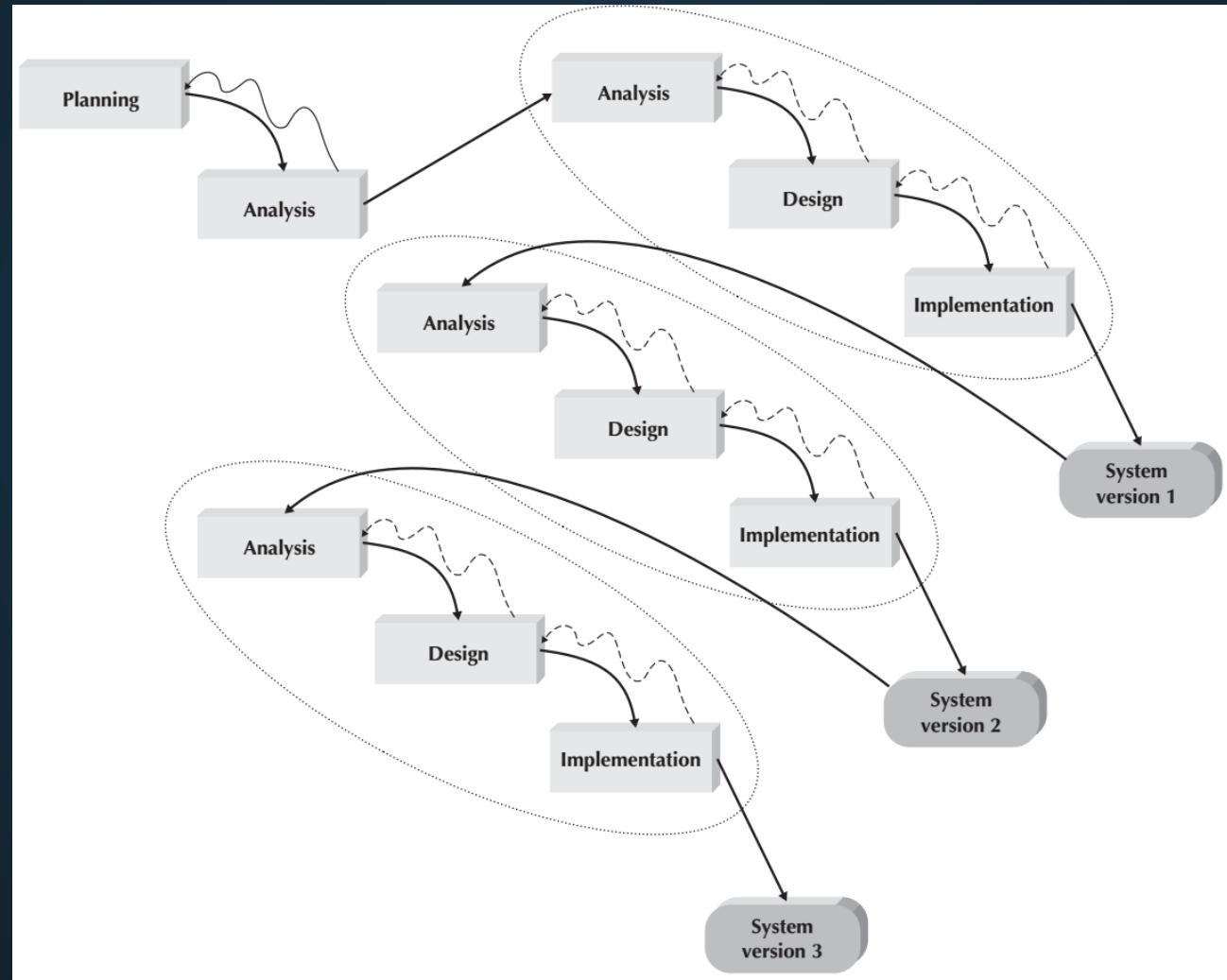
یک سیستم کلی را به مجموعه ای از نسخه هایی که به صورت متوالی توسعه می یابند، تجزیه می کند. مرحله تجزیه و تحلیل مفهوم کلی سیستم را شناسایی می کند و تیم پروژه، کاربران و حامیان سیستم، نیازمندی ها را در یک سری نسخه طبقه بندی می کند.

Phased Development(I)

- Breaks an **overall system** into a series of ***versions*** that are developed **sequentially**. The analysis phase identifies the **overall system concept**, and the project team, users, and system sponsor then **categorize the requirements** into a series of **versions**.
- The **most important and fundamental requirements** are bundled into the **first version** of the system.
- The **analysis phase** then leads into design and implementation—but only with **the set of requirements identified** for **version 1**. Once version 1 is implemented, work begins on version 2. **Additional analysis** is performed based on the previously identified requirements and combined with **new ideas and issues** that arose from the **users' experience** with version 1. Version 2 then is designed and implemented, and work immediately begins on the next version. This process continues until the system is complete or is no longer in use.



Phased Development(II)





Phased Development(III)

- ashkal umdeh ayin ast ke karpalan shrou b-h kar ba sistem hahi mi
- .knd ke umda naqsh hestnd
- shnasiyi mhem trin o mafidtrin wizrgi ha o گngandn anha dr nshh ool o
- mdiriyat anttzarat karpalan dr tol misir bsiyar mhem ast

- It has the advantage of **quickly** getting a **useful system** into the **hands of the users**.
- Although the system does not perform **all the functions** the users need at first, it does begin to provide **business value sooner** than if the system were delivered after completion.
- Likewise, because users begin to work with the system sooner, they are more likely to **identify important additional requirements sooner** than with structured design situations.
- The **major drawback** is that users begin to work with systems that are **intentionally incomplete**. It is critical to identify the most **important** and **useful** features and include them in the **first version** and to manage **users' expectations** along the way.

این مزیت این است که به سرعت یک سیستم مفید را در اختیار کاربران قرار می دهد.
اگرچه سیستم در ابتدا تمام عملکردهای مورد نیاز کاربران را انجام نمی دهد، اما زودتر از زمانی که سیستم پس از تکمیل تحویل داده شود، ارزش تجاری را ارائه می کند.
به همین ترتیب، از آنجایی که کاربران زودتر شروع به کار با سیستم می کنند، احتمال بیشتری دارد که نیازهای اضافی مهم را زودتر از موقعیت های طراحی ساختاریافته شناسایی کنند

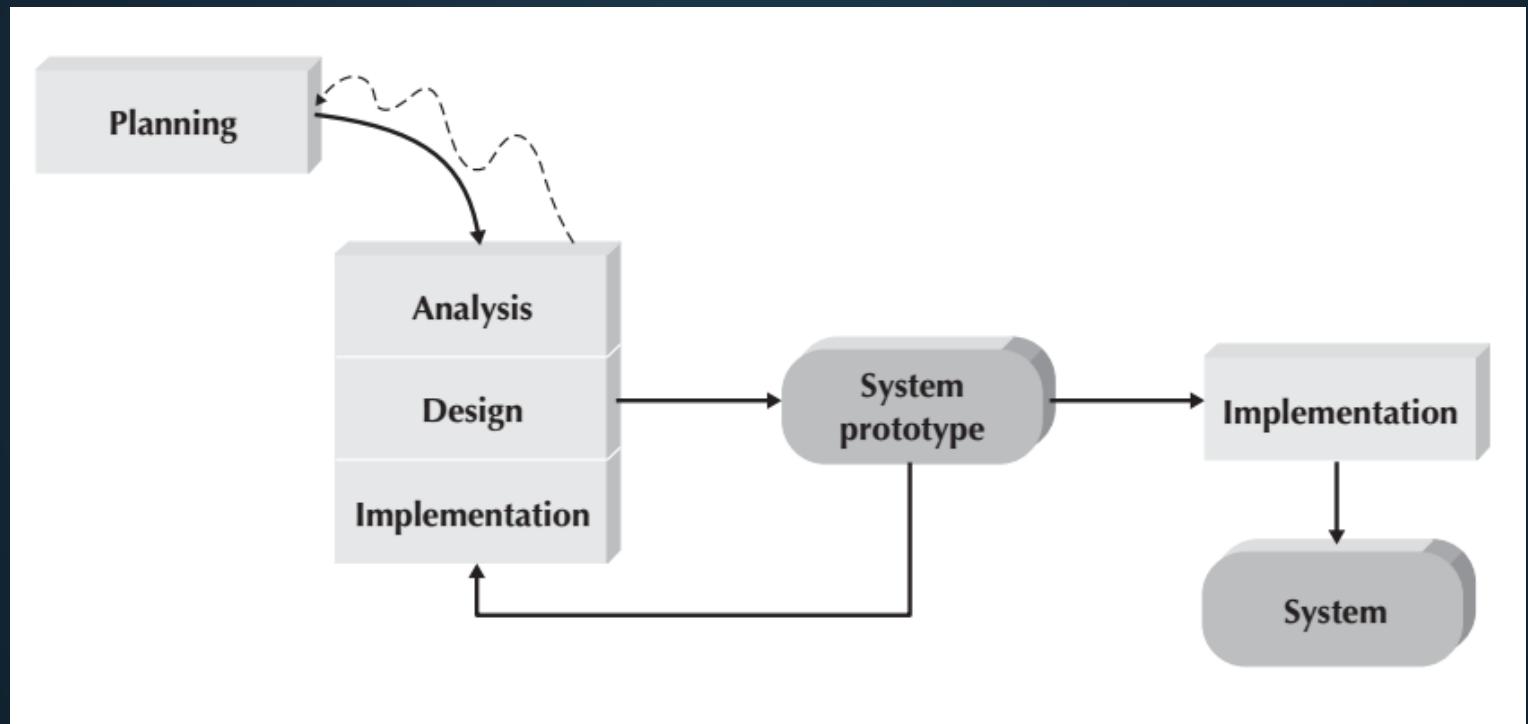


Prototyping(I)

- Performs the analysis, design, and implementation phases concurrently, and all three phases are performed repeatedly in a cycle until the system is completed.
- Basics of analysis and design are performed, and work immediately begins on a *system prototype*, a quick-and-dirty program that provides a minimal amount of features. The first prototype is usually the first part of the system that is used.
- This is shown to the users and the project sponsor, who provide comments. These comments are used to reanalyze, redesign, and re-implement a second prototype, which provides a few more features. This process continues in a cycle until the analysts, users, and sponsor agree that the prototype provides enough functionality to be installed and used in the organization.
- After the prototype (now called the “system”) is installed, refinement occurs until it is accepted as the new system.



Prototyping(II)





اغلب نمونه اولیه دستخوش تغییرات قابل توجهی می شود که بسیاری از تصمیمات اولیه طراحی ضعیف می شوند. این میتواند باعث ایجاد مشکلاتی در توسعه سیستمهای پیچیده شود، زیرا مسائل و مشکلات اساسی تازمانی که فرآیند توسعه به خوبی انجام نشود، شناسایی نمیشوند.

مزیت اصلی این است که بسیار سریع سیستمی را فراهم می کند که کاربران می توانند با آن تعامل داشته باشند، حتی اگر در ابتدا برای استفاده گسترشده سازمانی آماده نباشد. نمونه سازی به کاربران اطمینان می دهد که تیم پروژه بر روی سیستم کار می کند (تأخر طولانی وجود ندارد که کاربران پیشرفت کمی در آن مشاهده کنند)، و نمونه سازی به بهبود سریعتر نیازهای واقعی کمک می کند.

Prototyping(III)

- The **key advantage** is that it **very quickly** provides a system with which the users can **interact**, even **if it is not ready** for widespread organizational use at first.
- Prototyping reassures the users that the **project team is working on the system** (there are **no long delays** in which the users see little progress), and prototyping helps to **more quickly refine** real requirements.
- The **major problem** is that its **fast-paced system releases** challenge attempts to conduct **careful, methodical analysis**. Often the prototype undergoes such significant changes that many **initial design decisions** become **poor** ones. This can cause problems in the development of **complex** systems because **fundamental issues** and problems **are not recognized** until well into the development process.



Throwaway Prototyping(I)

هر یک از این مسائل با تجزیه و تحلیل، طراحی و ساخت یک نمونه اولیه طراحی بررسی می شود.

- These prototypes are used for a **very different purpose** than those previously discussed, and they have a **very different appearance**.
- It has a relatively thorough analysis phase that is used to gather information and to develop ideas for the system concept. However, users might **not** completely **understand** many of the **features** they suggest, and there may be challenging technical issues to be solved. Each of these **issues** is examined by **analyzing**, **designing**, and **building** a ***design prototype***. A design prototype is not a working system; it is a **product that represents a part of the system** that needs additional refinement, and it contains only enough detail to **enable users to understand** the issues under consideration.

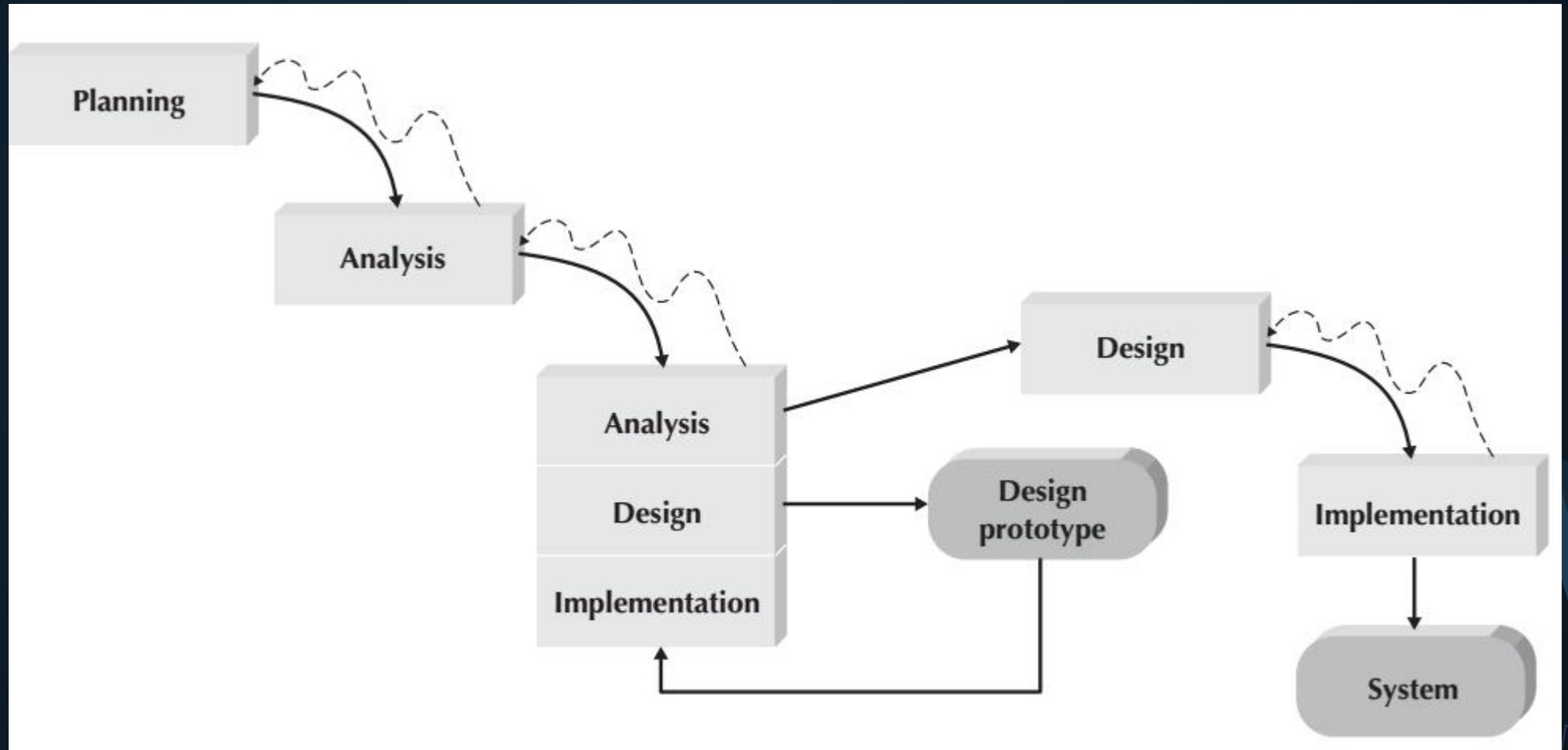


Throwaway Prototyping(II)

- Each of the prototypes is used to minimize the risk associated with the system by confirming that important issues are understood before the real system is built.
- Once the issues are resolved, the project moves into design and implementation. At this point, the design prototypes are thrown away, which is an important difference between these methodologies and prototyping methodologies, in which the prototypes evolve into the final system.
- It can take longer to deliver the final system as compared to prototyping-based methodologies, but produces more stable and reliable systems.



Throwaway Prototyping(III)





Agile

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

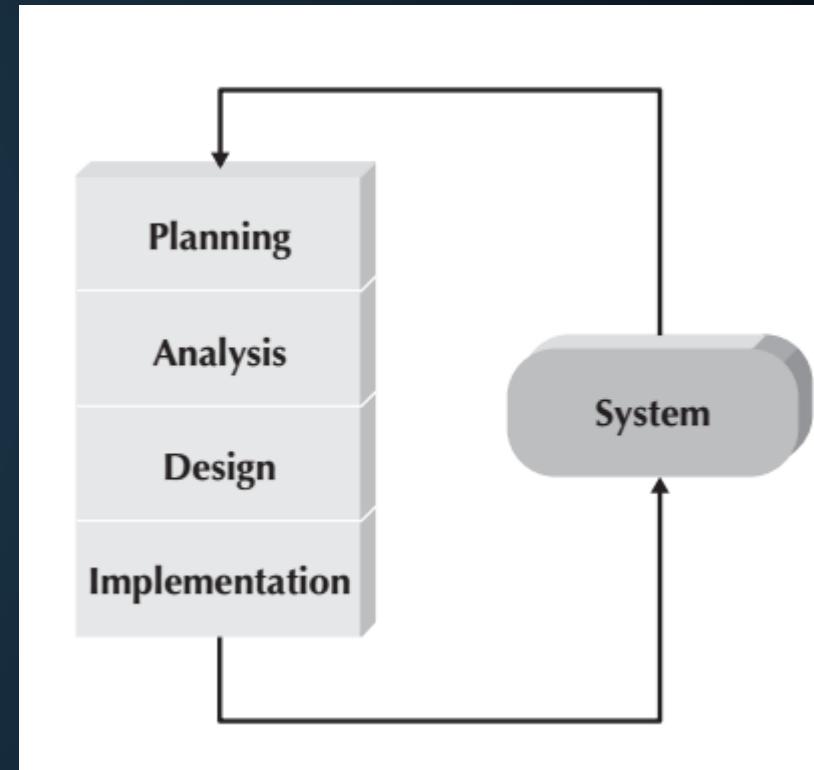
Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.





Selecting the Appropriate Development Methodology

Ability to Develop Systems	Structured Methodologies		RAD Methodologies			Agile Methodologies	
	Waterfall	Parallel	Phased	Prototyping	Throwaway Prototyping	XP	SCRUM
With Unclear User Requirements	Poor	Poor	Good	Excellent	Excellent	Excellent	Excellent
With Unfamiliar Technology	Poor	Poor	Good	Poor	Excellent	Good	Good
That Are Complex	Good	Good	Good	Poor	Excellent	Good	Good
That Are Reliable	Good	Good	Good	Poor	Excellent	Excellent	Excellent
With a Short Time Schedule	Poor	Good	Excellent	Excellent	Good	Excellent	Excellent
With Schedule Visibility	Poor	Poor	Excellent	Excellent	Good	Excellent	Excellent



Software Development Methodology(SDM)

چارچوبی برای به کارگیری شیوه های مهندسی نرم افزار با هدف
خاص ارائه ابزار لازم برای توسعه سیستم های فشرده نرم افزار

- A **framework** for applying software engineering practices with the specific aim of providing the necessary means for developing software-intensive systems.
- Have two parts.
 1. A set of **modeling conventions** comprising a Modeling **Language** (syntax and semantics)
 2. A **Process**, which
 - provides guidance as to the **order of the activities**,
 - specifies **what artifacts** should be developed using the Modeling Language,
 - directs the **tasks** of individual developers and the team as a whole,
 - offers **criteria** for monitoring and measuring a project's **products** and **activities**.

یک فرآیند، که در مورد ترتیب فعالیت ها راهنمایی می کند ■، مشخص می کند که چه مصنوعاتی باید با استفاده از زبان مدل سازی توسعه داده شوند ■، وظایف توسعه دهنگان فردی و تیم را به عنوان یک کل هدایت می کند ■، معیارهایی برای نظارت و اندازه گیری محصولات و فعالیت های یک پروژه ارائه می دهد ■

مجموعه ای از قراردادهای مدل سازی شامل یک زبان مدل سازی
(نحو و معناشناسی)



Unified Modelling Language (UML)

هر توسعه دهنده متداول‌لوژی و نشانه گذاری خاص خود را داشت

ارائه یک واژگان مشترک از اصطلاحات شی گرا و UML هدف تکنیک های نموداری به اندازه کافی غنی برای مدل سازی هر پروژه توسعه سیستمی از تجزیه و تحلیل تا پیاده سازی بود

- Each developer had his or her own methodology and notation.
- A standard set of diagramming techniques, *Unified Modeling Language(UML)*.
- The objective of UML was to provide a common vocabulary of object-oriented terms and diagramming techniques rich enough to model any systems development project from analysis through implementation.



References

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



What we will talk about next...

- Object-oriented principles
- RUP
- Scrum



Software Engineering I

Object-Oriented Principles

Dr. Elham Mahmoudzadeh
Isfahan University of Technology
mahmoudzadeh@iut.ac.ir
2022



Object-oriented systems

تمرکز بر ثبت ساختار و رفتار سیستمهای اطلاعاتی در مأموریت کوچکی که هم دادهها و هم فرآیند را در بر میگیرد، به نام اشیا.

- Focus on capturing the **structure** and **behavior** of information systems in **little modules** that encompass both **data** and **process**, called *objects*.
- A **class** is the **general template** we use to define and create specific instances, or objects.
- Every object is associated with a class.



Three parts of an object

- **Attributes**: describe **information** about the **object**.
- **Behavior**: specify **what** the object **can do**.
- **State**: defined by the value of **its attributes** and **its relationships** with other objects at a particular point in **time**.



Principles

- Abstraction
- Encapsulation
- Modularity
- Inheritance



Abstraction

- Is the process of taking away or **removing characteristics** from something in order to reduce it to a set of **essential characteristics**.
- When you face with the problem, **look at the most important things**.
- **Reduce the complexity** to understand. Then, refine the problem and focus on the **second level** of the features.



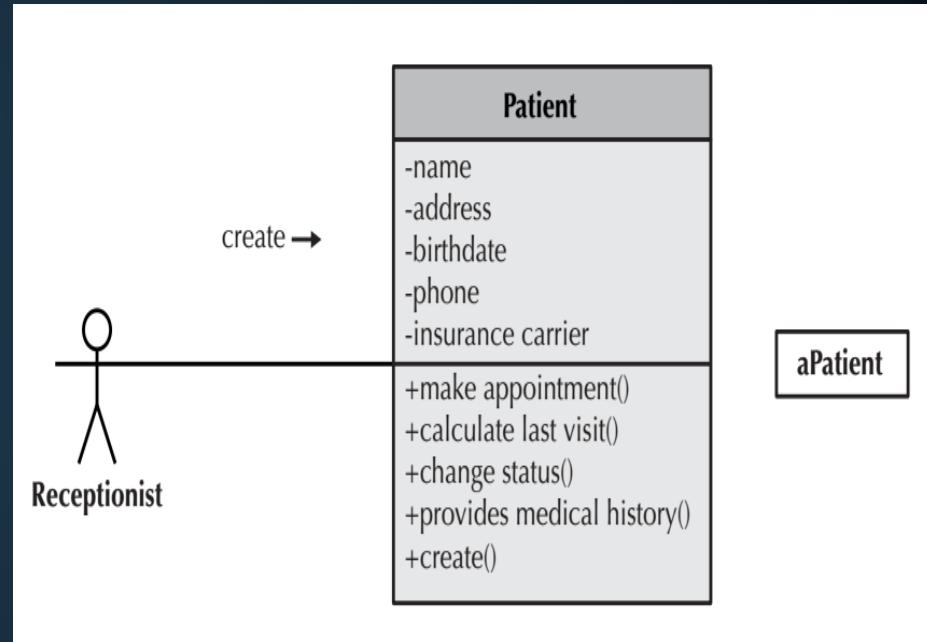
Encapsulation

- Is the **combination of process and data** into a **single entity**.
- See the class as a **black box**.
- **Information** required to be passed to the module and the **information returned** from the module are **published**.
- Exactly how the module implements the required functionality is **not relevant**. We really **do not care how** the object **performs** its functions, as long as the functions occur.
- It is used to **hide** the **internal representation** of an **object** from the outside.
- You cannot **access** to data of a class directly, but try to **request** to the class.
- **Access level** is very important.



Encapsulation(Cnt'd)

- The fact that we can **use** an **object** by calling **methods** is the key to **reusability** because it **shields** the internal workings of the object **from changes** in the outside system, and it keeps the system from **being affected** when changes are made to an object.
- The only information that an **object** needs to know is the **set of operations**, or **methods**, that other objects can perform and **what messages** need to be sent to trigger them.
 - Messages*** are information sent to objects to trigger methods.





Modularity

- Modularity is the degree to which a system's components are made up of relatively **independent components** or parts which can be **combined**.
- **Decompose** a system into the objects that are **loosely coupled** with each other, connection should be as weak as possible.
- Objects should be **highly cohesive**, it is better for an **object** to be a **single minded entity**.

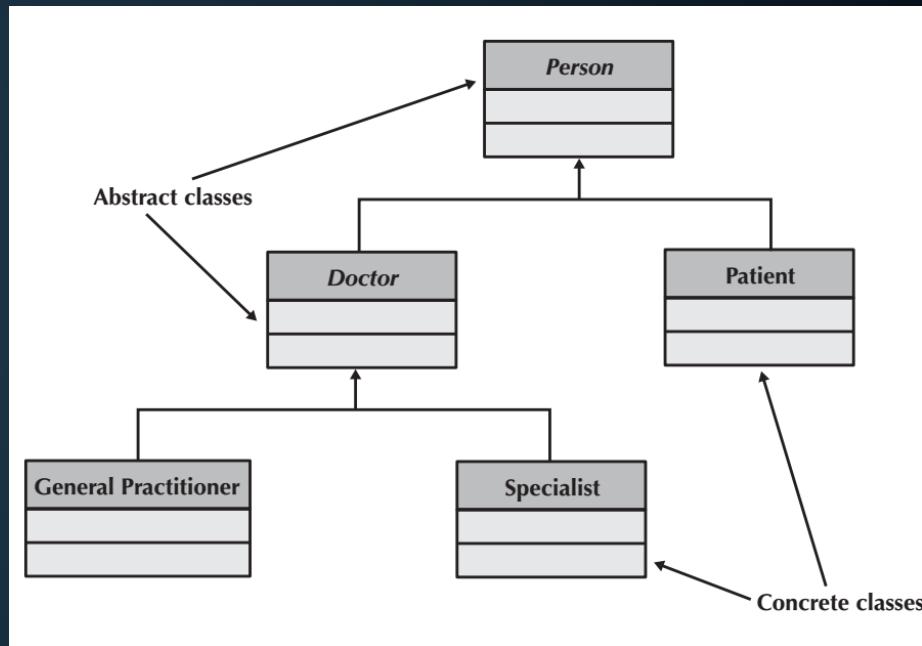


ماژولاریت درجه ای است که اجزای یک سیستم از اجزا یا قطعات نسبتاً مستقل تشکیل شده اند که می توانند با هم ترکیب شوند.
یک سیستم را به اشیایی که به طور سست با یکدیگر جفت شده اند تجزیه ●
کنید، اتصال باید تا حد امکان ضعیف باشد
● اشیا باید بسیار منسجم باشند، بهتر است یک شی یک موجودیت واحد باشد



Inheritance

- Common sets of attributes and methods can be organized into *super classes*.
- Try to inherit data and operation from *superclass*.
- Makes it simpler to define classes. Instead of repeating the attributes and methods in the *subclasses*, the attributes and methods that are common are placed in *superclass* and inherited by the classes below it.





Benefits of Object-Oriented Systems Analysis and Design

- Concepts in the object-oriented approach enable analysts to **break a complex system** into **smaller**, **more-manageable modules**, work on the modules **individually**, and easily piece the modules back together to form an information system.
- The **modularity** makes systems development **easier to grasp**, easier to **share** among members of a **project team**, and easier to **communicate** to **users**, who are needed to provide **requirements** and confirm **how well** the system meets the requirements throughout the systems development process.
- By modularizing, the project team actually is creating **reusable pieces** that can be plugged into **other systems** or used as **starting points** for other projects. This can **save time** because new projects don't have to start completely from scratch.



References

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



What we will talk about next...

- Object-Oriented approach



Software Engineering I

Object-Oriented Approach

Dr. Elham Mahmoudzadeh
Isfahan University of Technology
mahmoudzadeh@iut.ac.ir
2022



Introduction

فرآیند تجزیه مسئله یا فرآیند محور یا داده محور است

چگونه یک مشکل تجزیه می شود

- The primary difference between a **traditional** approach and an **object-oriented** approach is **how a problem is decomposed**.
- In traditional approaches, the problem-decomposition process is either **process-centric** or **data-centric**.
 - Processes and data are **so closely** related that it is difficult to pick one or the other as the primary focus.
- *Object-oriented methodologies* attempt to **balance** the **emphasis** between process and data by focusing the decomposition of problems on **objects** that **contain** both **data** and **processes**.



Modern object-oriented approach for developing information systems

- Use-case driven,
 - Architecture-centric,
 - Iterative and incremental.
- مورد استفاده،
 - معماری محور،
 - تکراری و افزایشی



- *Use case* is the primary modeling tool.
- A use case describes **how the user interacts** with the system to perform some activity, such as placing an order, making a reservation, or searching for information.
- A use case is used to **identify** and to **communicate** the **requirements** for the system to the **programmers** who must write the system.
- Also, use case is used for **testing**.



2- Architecture-Centric

- Any modern approach to systems analysis and design should be architecture-centric.
- Support at least three separate but interrelated architectural views of a system: *functional, structural, and behavioral*.

هر رویکرد مدرن برای تجزیه و تحلیل و طراحی سیستم ها باید معماری محور باشد.
حداقل از سه نمای معماری مجزا اما مرتبط با یک سیستم پشتیبانی کنید.



Three views of the system

- The *functional*, or *external* view: describes the behavior of the system from the perspective of the user.
- The *structural*, or *static* view: describes the system in terms of *attributes*, *methods*, *classes*, and *relationships*.
- The *behavioral*, or *dynamic* view: describes the behavior of the system in terms of messages passed among *objects* and *state changes* within an object.



3- Iterative and Incremental

در طول عمر پروژه تحت آزمایش و اصلاح
مستمر قرار می گیرد.

- Modern object-oriented systems analysis and design approaches emphasize *iterative* and *incremental* development that undergoes continuous testing and refinement throughout the life of the project.
- This implies that the systems analysts develop their understanding of a user's problem by building up the three architectural views little by little.



Iterative Development (I)

- Is a **planned rework** strategy.
- We use **multiple passes** to **improve** what we are building so we can converge on a good solution.
- Is an excellent way to **improve the product** as it is being developed.
- The **biggest downside** is that in the presence of **uncertainty** it can be difficult up front to determine **(plan)** how many improvement passes will be necessary.



Iterative Development (II)

- For example, we might start by creating a **prototype** to acquire **important knowledge** about a poorly known piece of the product. Then we might create a **revised version** that is somewhat better, which might in turn be followed by a pretty good version.
- In the course of writing this book, for example, I **wrote** and **rewrote** each of the chapters several times as I **received feedback** and as my **understanding** of how I wanted to communicate a topic improved.



Incremental Development (I)

- Based on the age-old principle of “Build some of it before you build all of it.”
- We avoid having one large, big-bang-style at the end of development.
- Instead, we break the product into smaller pieces so that we can build some of it, learn how each piece is to survive in the environment in which it must exist, adapt based on what we learn, and then build more of it.



Incremental Development (II)

عملکرد سیستم را به چند قسمت (بخش) تقسیم می کند
اطلاعات مهمی به ما مطبیق دهیم و نحوه ادامه کار را
تغییر دهیم.

- Slices the system **functionality** into **increments (portions)**.
- Gives us important information that allows us to **adapt** our **development** effort and to change how we proceed.
- The **biggest drawback** to is that by **building in pieces**, we risk **missing** the **big picture** (we see the trees but not the forest).



Incremental Development (III)

- For example, while writing this book, I wrote a **chapter at a time** and sent each chapter out for review as it was completed, rather than trying to receive feedback on the entire book at once.
- This gave me the opportunity to **incorporate that feedback** into future chapters, adjusting my tone, style, or delivery as needed.
- It also gave me the opportunity to **learn incrementally** and apply what I learned from **earlier chapters** to later chapters.



Iterative and Incremental

- The systems analyst does this by working **with the user** to create a **functional representation** of the system under study.
- Next, the analyst attempts to build a **structural representation** of the evolving system. Using the structural representation of the system, the analyst distributes the functionality of the system over the evolving structure to create a **behavioral representation** of the evolving system.
- As an **analyst works with the user** in developing the three architectural views of the evolving system, the **analyst iterates** over each of and among the views.
- That is, as the analyst better **understands** the **structural** and **behavioral** views, the analyst uncovers **missing requirements** or misrepresentations in the **functional** view.

به این معنا که وقتی تحلیلگر دیدگاه های ساختاری و رفتاری را بهتر درک می کند تحلیلگر الزامات گمشده یا ارائه نادرست در نمای عملکردی را کشف می کند.



References

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



What we will talk about next...

- RUP
- Scrum

Software Engineering I

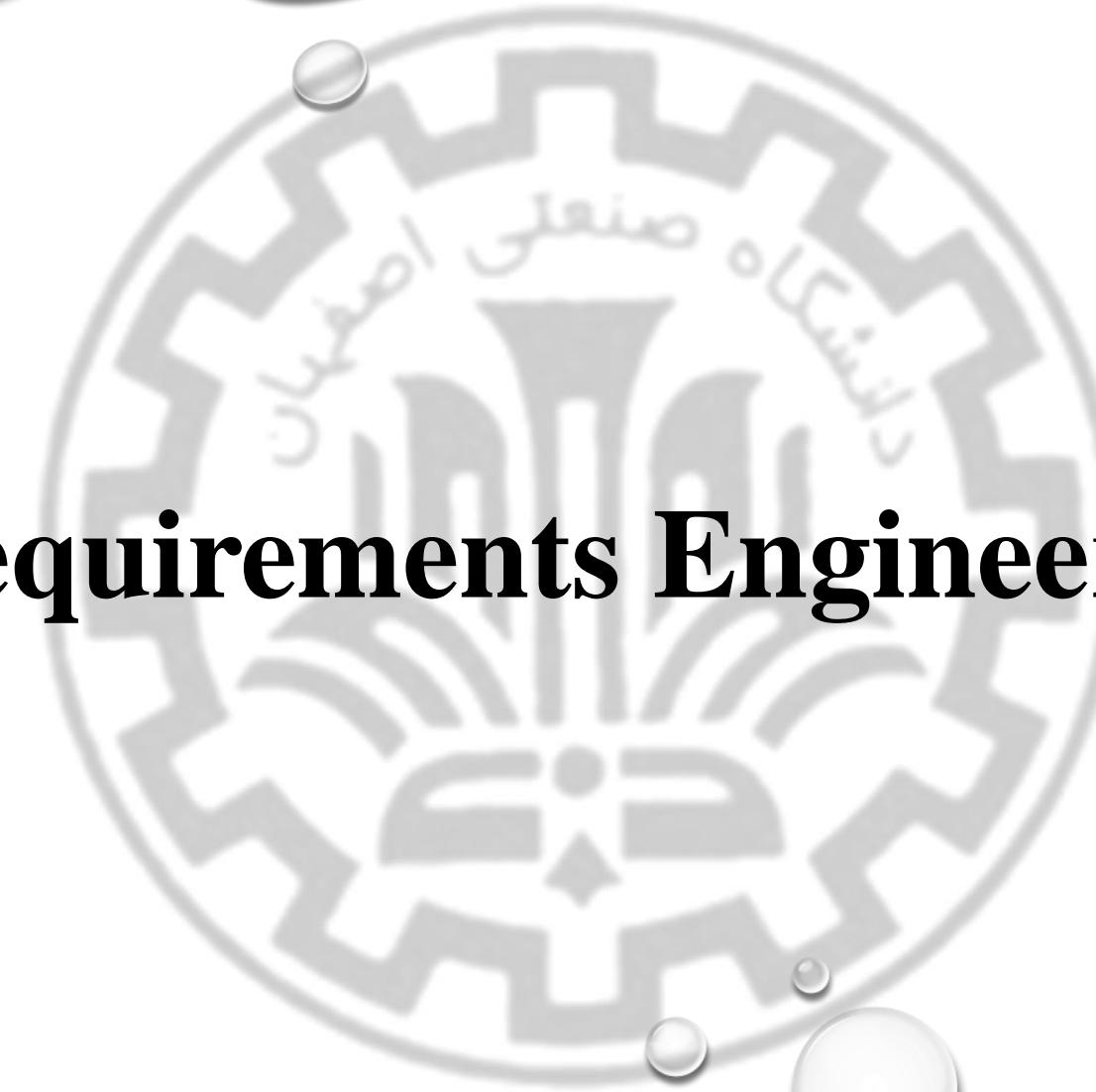
Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2022

Requirements Engineering



Software Development Life Cycle

- Planning
- Analysis
- Design
- Implementation

Introduction

- The systems development process aids an organization in moving from the current system (often called the *as-is system*) to the new system (often called the *to-be system*).
- The output of planning, is the system request, which provides general ideas for the to-be system, defines the project's scope, and provides the initial work-plan.
- Analysis takes the general ideas in the system request and refines them into a detailed requirements definition.



Let's Start

Steps(I)

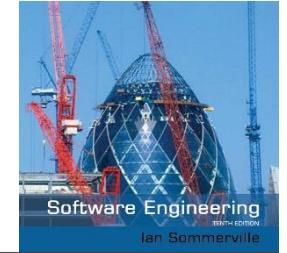
1. Preparing proposal
2. Requirements determination
 - User story
3. Abstract Business Process Modelling
4. Analysis
 - Functional Modelling
 - Structural Modelling
 - Behavioral Modelling

Steps(II)

5. Design

- Optimization
- Database Management
- User Interface
- Physical Architecture

Requirements engineering



- ❖ The process of establishing the **services** that a customer **requires** from a system and the **constraints** under which it operates and is developed.

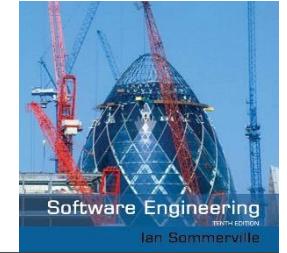
فرآیند ایجاد خدماتی که مشتری از یک سیستم نیاز دارد و محدودیت هایی که تحت آن عمل می کند و توسعه می یابد.

What is requirement?

- Requirement is : **new system's capabilities.**
- A *requirement* is simply a statement of **what the system must do** or **what characteristic** it must have.
- During analysis, requirements are written from the **perspective** of the **businessperson**, and they focus on the "**what**" of the system.
 - Focus on the needs of the business user, **business requirements** (user requirements).

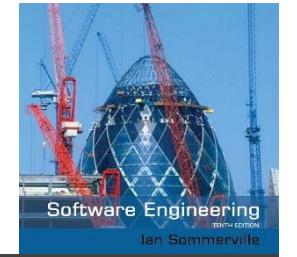
مورد نیاز این است: قابلیت های سیستم جدید •
یک الزام صرفاً بیانی است از آنچه که سیستم باید انجام دهد یا چه ویژگی هایی •
باید داشته باشد.
در طول تجزیه و تحلیل، الزامات از دیدگاه تاجر نوشته می شود، و آنها بر "چه •
چیزی" سیستم تمرکز می کنند
(تمرکز بر نیازهای کاربر تجاری، الزامات تجاری (نیازهای کاربر •

System stakeholders



- ✧ Any person or organization who is affected by the system in some way and so who has a legitimate interest
- ✧ Stakeholder types
 - End users
 - System managers
 - System owners
 - External stakeholders

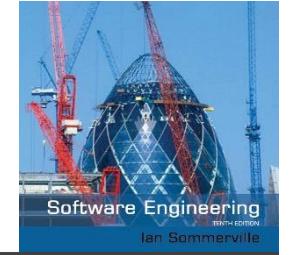
هر شخص یا سازمانی که به نحوی تحت تأثیر سیستم قرار گرفته و دارای منافع
مشروع باشد.
انواع ذینفعان
کاربران نهایی
مدیران سیستم
صاحبان سیستم
ذینفعان خارجی



Agile methods and requirements

- ✧ Many agile methods argue that producing **detailed system requirements** is a **waste of time** as requirements **change** so quickly.
- ✧ The **requirements document** is therefore always **out of date**.
- ✧ **Agile** methods usually use **incremental requirements engineering** and may express requirements as '**user stories**'.
- ✧ This is practical for business systems but problematic for systems that require **pre-delivery analysis** (e.g. **critical systems**) or systems developed by **several teams**.

بسیاری از روش‌های چاک استدلال می‌کنند که تولید دقیق نیازمندی‌های سیستم اتلاف وقت است زیرا نیازمندی‌ها به سرعت تغییر می‌کنند. بنابراین سند الزامات همیشه قدیمی است. روش‌های چاک معمولاً از مهندسی نیازمندی‌های افزایشی استفاده می‌کنند و ممکن است الزامات را به عنوان داستان کاربر «بیان کنند». این برای سیستم‌های تجاری عملی است اما برای سیستم‌هایی که نیاز به تجزیه و تحلیل قبل از تحویل دارند (مانند سیستم‌های حیاتی) یا سیستم‌های توسعه یافته توسط چندین تیم مشکل ساز است.



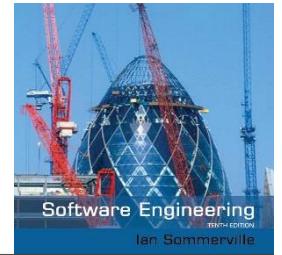
Functional and non-functional requirements

Functional Requirements

- Relates directly to a process that a system has to perform or information it needs to contain.
 - For example, requirements stating that a system must have the ability to search for a product.
- Flow directly into the creation of functional, structural, and behavioral models that represent the functionality of the evolving system.

الزامات عملکردی
به طور مستقیم به فرآیندی که یک سیستم باید انجام دهد یا اطلاعاتی که باید حاوی آن باشد •
مریبوط می شود.
به عنوان مثال، الزاماتی مبنی بر اینکه یک سیستم باید توانایی جستجوی یک محصول را داشته باشد •
مستقیماً در ایجاد مدل های عملکردی، ساختاری و رفتاری جریان می یابد که نشان دهنده •
عملکرد سیستم در حال تکامل است.

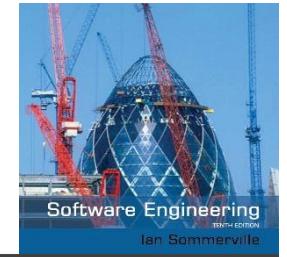
Functional requirements



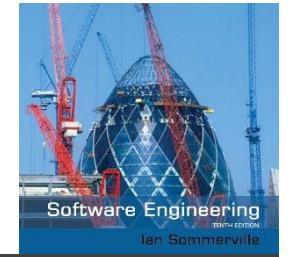
- ✧ Describe functionality or system services.
- ✧ Depend on the type of software, expected users and the type of system where the software is used.
- ✧ Functional user requirements may be high-level statements of what the system should do.
- ✧ Functional system requirements should describe the system services in detail.

عملکرد یا خدمات سیستم را شرح دهید.
به نوع نرم افزار، کاربران مورد انتظار و نوع سیستمی که نرم افزار در آن استفاده می شود بستگی دارد.
الزامات کاربردی کاربر ممکن است بیانیه های سطح بالایی از آنچه سیستم باید انجام دهد باشد.
الزامات سیستم عملکردی باید خدمات سیستم را با جزئیات توصیف کند.

Mentcare system: functional requirements



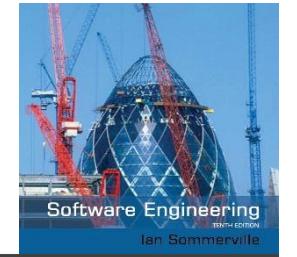
- ✧ A user shall be able to search the appointments lists for all clinics.
- ✧ The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
- ✧ Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.



Requirements imprecision

- ✧ Problems arise when functional requirements are not precisely stated.
- ✧ Ambiguous requirements may be interpreted in different ways by developers and users.
- ✧ Consider the term ‘search’ in requirement 1
 - User intention – search for a patient name across all appointments in all clinics;
 - Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then search.

مشکلات زمانی به وجود می آیند که الزامات عملکردی به طور دقیق بیان نشده باشند.
الزامات مهم ممکن است به روش های مختلفی توسط توسعه دهندهان و کاربران تفسیر شود.



Requirements completeness and consistency

- ✧ In principle, requirements should be both **complete** and **consistent**.
- ✧ Complete
 - They should include **descriptions** of **all facilities** required.
- ✧ Consistent
 - There should be **no conflicts** or **contradictions** in the descriptions of the system facilities.
- ✧ **In practice**, because of system and environmental complexity, it is **impossible** to produce a complete and consistent requirements document.

کامل آنها باید شامل توضیحات تمام امکانات مورد نیاز باشند.
سازگار هیچ گونه تضاد یا تناقضی در توضیحات امکانات سیستم وجود نداشته باشد
در عمل، به دلیل پیچیدگی سیستم و محیطی، تولید یک سند الزامات کامل و منسجم
غیرممکن است.

Non-functional Requirements

- Refer to behavioral properties that the system must have, such as performance and usability.
 - The ability to access the system using a Web browser is considered a nonfunctional requirement.
- Can influence the rest of analysis (functional, structural, and behavioral models) but often do so only indirectly;
- Are used primarily in design when decisions are made about the database, the user interface, the hardware and software, and the system's underlying physical architecture.

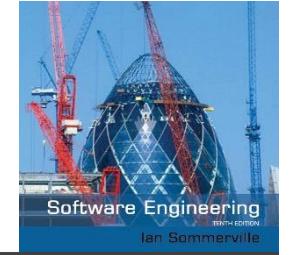
به ویژگی های رفتاری که سیستم باید داشته باشد، مانند عملکرد و قابلیت استفاده اشاره دارد.

امکان دسترسی به سیستم با استفاده از مرورگر وب یک نیاز غیر کاربردی در نظر گرفته می شود.

می تواند بر بقیه تحلیل ها (مدل های عملکردی، ساختاری و رفتاری) تأثیر بگذارد.

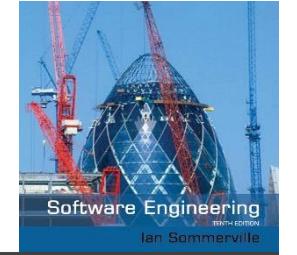
اما اغلب این کار را فقط به صورت غیر مستقیم انجام می دهد.

زمانی که تصمیماتی در مورد پایگاه داده، رابط کاربری، سخت افزار و نرم افزار و معماری فیزیکی زیربنایی سیستم گرفته می شود، عمدتاً در طراحی استفاده می شوند.



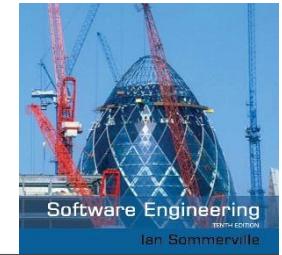
Non-functional requirements

- ❖ These define **system properties** and **constraints** e.g. **reliability**, **response time** and **storage** requirements. Constraints are **I/O device capability**, **system representations**, etc.
- ❖ Non-functional requirements may be **more critical** than functional requirements. If these are **not met**, the system may be **useless**.



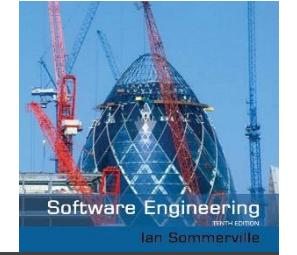
Non-functional requirements **implementation**

- ✧ Non-functional requirements may affect the **overall architecture** of a system rather than the **individual components**.
 - For example, to ensure that **performance** requirements are met, you may have to **organize the system** to **minimize** **communications** between components.

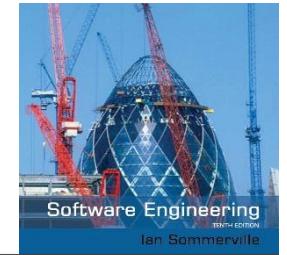


Metrics for specifying nonfunctional requirements

Property	Measure	
Speed	Processed transactions/second User/event response time Screen refresh time	تراکنش های پردازش شده/ثانیه زمان پاسخ کاربر/رویداد زمان تازه کردن صفحه نمایش
Size	Mbytes Number of ROM chips	
Ease of use	Training time Number of help frames	
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability	احتمال در دسترس نبودن میزان وقوع شکست دسترسی
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure	زمان راه اندازی مجدد پس از شکست درصد رویدادهایی که باعث شکست می شوند احتمال خراب شدن داده ها در صورت شکست
Portability	Percentage of target dependent statements Number of target systems	درصد عبارات وابسته به هدف تعداد سیستم های هدف



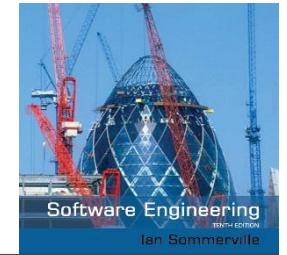
Requirements engineering processes



Requirements engineering processes

- ✧ The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- ✧ However, there are a number of generic activities common to all processes
 - Requirements elicitation;
 - Requirements analysis;
 - Requirements validation;
 - Requirements management.
- ✧ In practice, RE is an iterative activity in which these processes are interleaved.

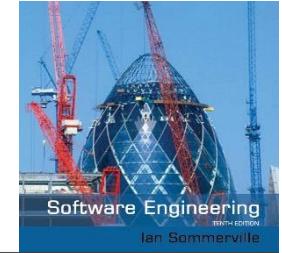
استخراج نیازمندی ها؛
تجزیه و تحلیل نیازمندی ها؛
اعتبارسنجی الزامات؛
مدیریت نیازمندی ها



Requirements elicitation and analysis

- ✧ Sometimes called requirements elicitation or requirements discovery.
- ✧ Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- ✧ May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders*.

کارشناسان حوزه، اتحادیه های کارگری

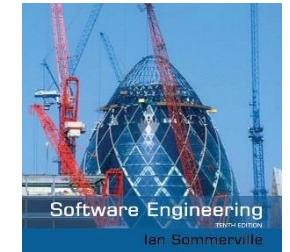


Requirements elicitation

- ✧ Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.
- ✧ Stages include:
 - Requirements discovery,
 - Requirements classification and organization,
 - Requirements prioritization and negotiation,
 - Requirements specification.

کشف نیازمندی ها،
طبقه بندی و سازماندهی نیازمندی ها،
اولویت بندی نیازمندی ها و مذاکره
مشخصات مورد نیاز.

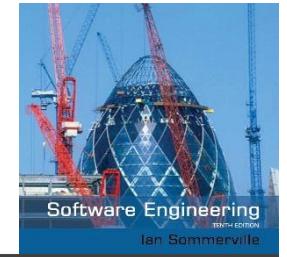
Problems of requirements elicitation



- ✧ Stakeholders don't know what they really want.
- ✧ Stakeholders express requirements in their own terms.
- ✧ Different stakeholders may have conflicting requirements.
- ✧ Organisational and political factors may influence the system requirements.
- ✧ The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.

ذینفعان نمی دانند واقعاً چه می خواهند.
ذینفعان الزامات خود را بیان می کنند.
ذینفعان مختلف ممکن است الزامات متقاضی داشته باشد.
عوامل سازمانی و سیاسی ممکن است بر الزامات سیستم تأثیر بگذارند.
الزامات در طول فرآیند تحلیل تغییر می کند. ممکن است ذینفعان جدیدی ظهور کنند و محیط کسب و کار ممکن است تغییر کند.

Process activities



✧ Requirements discovery

- Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.

✧ Requirements classification and organisation

- Groups related requirements and organises them into coherent clusters.

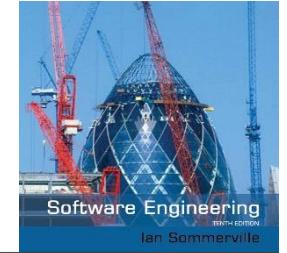
✧ Prioritisation and negotiation

- Prioritising requirements and resolving requirements conflicts.

✧ Requirements specification

- Requirements are documented and input into the next round of the spiral.

Requirements discovery



- ✧ The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.
- ✧ Interaction is with system stakeholders from managers to external regulators.
- ✧ Systems normally have a range of stakeholders.

Requirements Determination(I)

- Usually, users don't know exactly what they want, and analysts need to help them discover their needs.
- Analysts guide the users in explaining what is wanted from a system.
- Analysts help users critically examine the current state of systems and processes (the *as-is system*), identify exactly what needs to change, and develop a concept for a new system (the *to-be system*).

معمولًا کاربران دقیقاً نمی‌دانند چه می‌خواهند و تحلیلگران باید به آنها کمک کنند تا نیازهای خود را کشف کنند.
تحلیلگران کاربران را در توضیح آنچه از یک سیستم می‌خواهند راهنمایی می‌کنند.
تحلیلگران به کاربران کمک می‌کنند تا وضعیت فعلی سیستم‌ها و فرآیندها (سیستم همانطور که هست) را به طور انتقادی بررسی کنند، دقیقاً آنچه را که باید تغییر کند شناسایی کرده و مفهومی برای توسعه دهنده (tobe) یک سیستم جدید (سیستم

Requirements Determination(II)

- Creating a requirements definition is an iterative and ongoing process whereby the analyst collects information with requirements-gathering techniques.
- Then analyst analyzes the information to identify appropriate business requirements for the system.
- The requirements definition is kept up to date so that the project team and business users can refer to it and get a clear understanding of the new system.

ایجاد تعریف نیازمندی ها یک فرآیند تکراری و مداوم است که در آن تحلیلگر اطلاعات را با تکنیک های جمع آوری نیازمندی ها جمع آوری می کند.

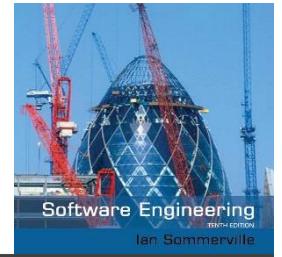
سپس تحلیلگر اطلاعات را برای شناسایی الزامات تجاری مناسب برای سیستم تجزیه و تحلیل می کند.

تعریف الزامات به روز نگه داشته می شود تا تیم پروژه و کاربران تجاری بتوانند به آن مراجعه کرده و درک روشنی از سیستم جدید بدست آورند.

Requirements gathering

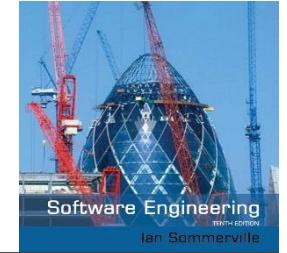
- Using a variety of techniques and make sure that the current business processes and the needs for the new system are well understood before moving into design.
- Not to discover later that they have wrong key requirements.
- All the key stakeholders must be included in the requirements-gathering process.

Interviewing

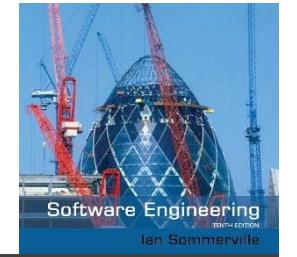


- ✧ Formal or informal interviews with stakeholders are part of most RE processes.
- ✧ Types of interview
 - Closed interviews based on pre-determined list of questions
 - Open interviews where various issues are explored with stakeholders.
- ✧ Effective interviewing
 - Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
 - Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

Interviews in practice



- ✧ Normally a mix of closed and open-ended interviewing.
- ✧ Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- ✧ Interviewers need to be open-minded without pre-conceived ideas of what the system should do
- ✧ You need to prompt the user to talk about the system by suggesting requirements rather than simply asking them what they want.



Problems with interviews

- ✧ Application specialists may use language to describe their work that isn't easy for the requirements engineer to understand.
- ✧ Interviews are not good for understanding domain requirements
 - Requirements engineers cannot understand specific domain terminology;
 - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating.

Requirements Gathering Techniques(I)

- **Interview:** is the **most commonly** used requirements-gathering technique. After all, it is natural—if you need to know something, you usually **ask someone.**
- **Joint Application Development (JAD):** is a technique that allows the **project team, users,** and **management** to **work together** to identify requirements for the system.
- **Questionnaires:** is a **set of written questions** used to obtain information from individuals.
 - Are often used when there is a **large number of people** from whom information and opinions are needed.

Requirements Gathering Techniques(II)

- **Document Analysis:** Project teams often use document analysis to understand the as-is system.
- **Observation:** the act of watching processes being performed, is a powerful tool for gathering information about the as-is system because it enables the analyst to see the reality of a situation, rather than listening to others describe it in interviews or JAD sessions. Observation is a good way to check the validity of information gathered from indirect sources such as interviews and questionnaires.

Which one is appropriate?

- No one technique is always better than the others.
- In practice most projects use a combination of techniques.
- It is important to understand the strengths and weaknesses of each technique and when to use.



	Interviews	Joint Application Design	Questionnaires	Document Analysis	Observation
Type of information	As-is, improvements, to-be	As-is, improvements, to-be	As-is, improvements	As-is	As-is
Depth of information	High	High	Medium	Low	Low
Breadth of information	Low	Medium	High	High	Low
Integration of information	Low	High	Low	Low	Low
User involvement	Medium	High	Low	Low	Low
Cost	Medium	Low to Medium	Low	Low	Low to Medium

Type of Information

- Different stages of the analysis process: *understanding the as-is system, identifying improvements, and developing the to-be system.*
- **Interviews** and **JAD** are commonly used in all three stages.
- In contrast, **document analysis** and **observation** usually are most helpful for *understanding the as-is*, although *occasionally* they provide information about current problems that need to *be improved*.
- **Questionnaires** are often used to gather information about the *as-is system* as well as *general information about improvements.*

Depth of Information

- Refers to **how rich and detailed** the information is that the technique usually produces and the extent to which the technique is useful for obtaining not only **facts and opinions** but also an **understanding of why** those facts and opinions **exist**.
- **Interviews** and **JAD** sessions are **very useful** for providing a **good depth** of rich and **detailed information** and helping the analyst to understand the **reasons behind them**.
- **Document** analysis and **observation** are useful for obtaining facts, but little beyond that.
- **Questionnaires** can provide a **medium depth** of information, soliciting both facts and opinions with **little understanding** of why they exist.

Breadth of Information

- Refers to the range of information and information sources that can be easily collected using the chosen technique.
- Questionnaires and document analysis are both easily capable of soliciting a wide range of information from a large number of information sources.
- Interviews and observation require the analyst to visit each information source individually and, therefore, take more time.
- JAD sessions are in the middle because many information sources are brought together at the same time.

به گستره ای از اطلاعات و منابع اطلاعاتی اشاره دارد که با استفاده از تکنیک انتخاب شده به راحتی می توان آنها را جمع آوری کرد

یکی از چالش برانگیزترین جنبه های جمع آوری نیازمندی ها، یکپارچه سازی اطلاعات از منابع مختلف است. به زبان ساده، افراد مختلف می توانند اطلاعات متنافضی ارائه دهند. ترکیب این اطلاعات و تلاش برای حل اختلاف در نظرات یا حقایق معمولاً بسیار زمان بر است زیرا به معنای تماس با هر منبع اطلاعاتی به نوبه خود، توضیح اختلاف و تلاش برای اصلاح اطلاعات است.

Integration of Information

- One of the most challenging aspects of requirements gathering is integrating the information from different sources.
- Simply put, different people can provide conflicting information. Combining this information and attempting to resolve differences in opinions or facts is usually very time consuming because it means contacting each information source in turn, explaining the discrepancy, and attempting to refine the information.
- All techniques suffer integration problems to some degree, but JAD sessions are designed to improve integration because all information is integrated when it is collected, not afterward. The immediate integration of information is the single most important benefit of JAD that distinguishes it from other techniques.

همه تکنیک ها تا حدی با مشکلات یکپارچه سازی مواجه هستند، اما برای بهبود یکپارچه سازی طراحی شده اند، زیرا همه JAD جلسات اطلاعات در هنگام جمع آوری یکپارچه می شوند، نه پس از آن. ادغام است که آن را از سایر JAD فوری اطلاعات تنها مهمترین مزیت تکنیک ها متمایز می کند

User Involvement

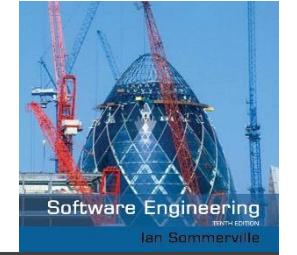
- Refers to the amount of time and energy the intended users of the new system must devote to the analysis process.
- It is generally agreed that as users become more involved, the chance of success increases.
- User involvement can have a significant cost, and not all users are willing to contribute valuable time and energy.
- Questionnaires, document analysis, and observation place the least burden on users.
- JAD sessions require the greatest effort.

به مقدار زمان و انرژی اشاره دارد که کاربران مورد نظر سیستم جدید باید به فرآیند تحلیل اختصاص دهند.
به طور کلی توافق بر این است که با مشارکت بیشتر کاربران، شانس موفقیت افزایش می یابد.
مشارکت کاربر می تواند هزینه قابل توجهی داشته باشد و همه کاربران مایل به مشارکت در زمان و انرژی ارزشمند نیستند.

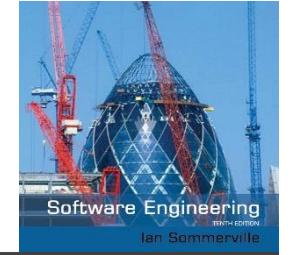
Cost

- Is always an important consideration.
- **Questionnaires, document analysis, and observation** are **low-cost techniques** (although observation can be quite time consuming).
- **Interviews and JAD sessions** generally have **moderate costs**. In general, **JAD sessions** are much **more expensive initially**, because they require many users to be absent from their offices for significant periods of time, and they often involve highly paid consultants. However, JAD sessions significantly **reduce the time spent in information integration** and thus can **cost less in the long term**.

معمولًاً هزینه های متوسطی دارند. به طور کلی، جلسات **JAD** مصاحبه ها و جلسات **JAD** در ابتدا بسیار گرانتر هستند، زیرا نیاز به غیبت بسیاری از کاربران برای مدت **JAD** زمان قابل توجهی در دفاتر خود دارند و اغلب مشاوران با حقوق بالایی را شامل می شوند. به طور قابل توجهی زمان صرف شده در یکپارچه سازی **JAD** با این حال، جلسات اطلاعات را کاهش می دهد و بنابراین می تواند در مدت هزینه کمتری داشته باشد.

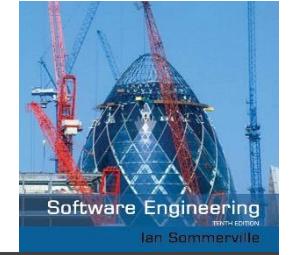


Requirements specification



Requirements specification

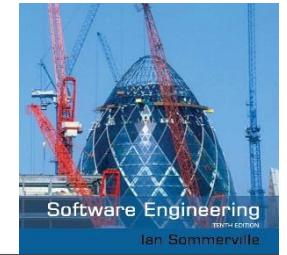
- ✧ The process of writing down the requirements in a requirements document.
- ✧ The requirements may be part of a contract for the system development
 - It is therefore important that these are as complete as possible.



Guidelines for writing requirements

- ✧ Invent a **standard** format and use it for all requirements.
- ✧ Use language in a **consistent** way. Use shall for mandatory requirements, should for desirable requirements.
- ✧ Use **text highlighting** to identify **key parts** of the requirement.
- ✧ Avoid the use of **computer jargon**.
- ✧ Include an **explanation** (rationale) of why a requirement is necessary.

یک فرمت استاندارد ابداع کنید و از آن برای همه نیازها استفاده کنید.
از زبان به شیوه ای ثابت استفاده کنید.
از پرجسته کردن متن برای شناسایی بخش های کلیدی مورد نیاز استفاده کنید.
از استفاده از اصطلاحات تخصصی کامپیوتري خودداری کنید.
توضیحی (منطقی) در مورد اینکه چرا یک نیاز ضروری است را بگنجانید.



Problems with natural language

✧ Lack of clarity

- Precision is difficult without making the document too long.

✧ Requirements confusion

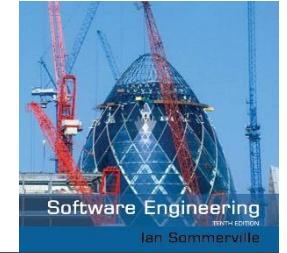
- Functional and non-functional requirements tend to be mixed-up.

✧ Requirements amalgamation

- Several different requirements may be expressed together.

عدم وضوح
دقت بدون خواندن سند دشوار است.
سردرگمی نیازها
الزامات عملکردی و غیر عملکردی معمولاً با هم مخلوط می شوند.
ادغام الزامات
چندین نیاز مختلف ممکن است با هم بیان شوند

Structured specifications



- ✧ An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
- ✧ This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.

، این برای برخی از انواع الزامات به خوبی کار می کند
مانند الزامات سیستم کنترل جاسازی شده، اما گاهی اوقات
برای نوشتن الزامات سیستم تجاری بسیار سفت و سخت
است.

Requirements document(I)

- Is a straightforward text report that simply lists the functional and nonfunctional requirements.
- Sometimes business requirements are prioritized on the requirements definition. They can be ranked as having high, medium, or low importance in the new system, or they can be labeled with the version of the system that will address the requirement (e.g., release 1). This practice is particularly important when using object-oriented methodologies since they deliver systems in an incremental manner.
- The most important purpose of the requirements definition, is to define the scope of the system.
- When discrepancies arise, the document serves as the place to clarify.

در صورت پروز اختلاف، سند به عنوان
مکانی برای روشن شدن عمل می کند

An Example

Nonfunctional Requirements

1. Operational Requirements

- 1.1. The system will operate in Windows environment.
- 1.2. The system should be able to connect to printers wirelessly.
- 1.3. The system should automatically back up at the end of each day.

2. Performance Requirements

- 2.1. The system will store a new appointment in 2 seconds or less.
- 2.2. The system will retrieve the daily appointment schedule in 2 seconds or less.

3. Security Requirements

- 3.1. Only doctors can set their availability.
- 3.2. Only a manager can produce a schedule.

4. Cultural and Political Requirements

- 4.1. No special cultural and political requirements are anticipated.

Functional Requirements

1. Manage Appointments

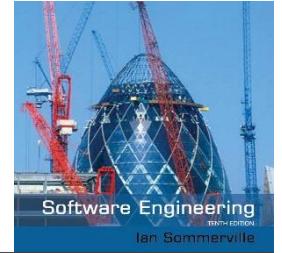
- 1.1. Patient makes new appointment.
- 1.2. Patient changes appointment.
- 1.3. Patient cancels appointment.

2. Produce Schedule

- 2.1. Office Manager checks daily schedule.
- 2.2. Office Manager prints daily schedule.

3. Record Doctor Availability

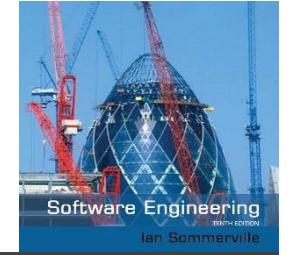
- 3.1. Doctor updates schedule



The software requirements document

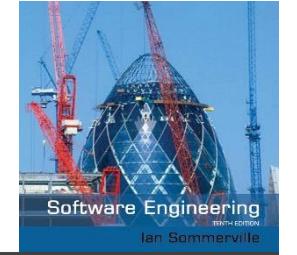
- ✧ The software requirements document is the **official statement** of **what is required** of the system developers.
- ✧ Should include both a definition of **user requirements** and a specification of the **system requirements**.
- ✧ It is NOT a design document. As far as possible, it should set of **WHAT** the system should do rather than **HOW** it should do it.

سند الزامات نرم افزار بیانیه رسمی آنچه از توسعه دهندگان سیستم مورد نیاز است است.
باید هم تعریفی از الزامات کاربر و هم مشخصات مورد نیاز سیستم را شامل شود.
این یک سند طراحی نیست. تا آنجا که ممکن است، باید آنچه را که سیستم باید انجام دهد به جای اینکه چگونه باید انجام دهد، تنظیم کند.



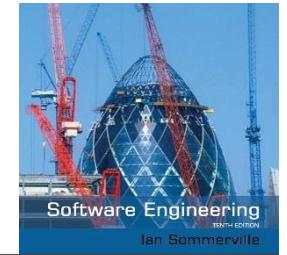
Requirements validation

Requirements validation



- ✧ Concerned with demonstrating that the requirements define the system that the customer **really wants**.
- ✧ Requirements error costs are high so validation is very important
 - Fixing a requirements error after delivery may cost up to **100 times the cost of fixing an implementation error**.

نگران این است که نشان دهد الزامات سیستمی را که مشتری واقعاً می خواهد تعریف می کند. هزینه های خطای الزامات بالاست، بنابراین اعتبارسنجی بسیار مهم است. رفع خطای الزامات پس از تحویل ممکن است تا 100 برابر هزینه رفع خطای پیاده سازی هزینه داشته باشد.

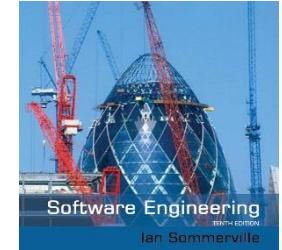


Requirements checking

- ✧ **Validity.** Does the system provide the functions which best support the customer's needs?
- ✧ **Consistency.** Are there any requirements conflicts?
- ✧ **Completeness.** Are all functions required by the customer included?
- ✧ **Realism.** Can the requirements be implemented given available budget and technology
- ✧ **Verifiability.** Can the requirements be checked?

اعتبار.
آیا سیستم عملکردهایی را ارائه می دهد که به بهترین وجه نیازهای مشتری را پشتیبانی می کند؟
سازگاری
آیا تعارض الزامات وجود دارد؟
کامل بودن
آیا تمام عملکردهای مورد نیاز مشتری شامل می شود؟
واقع گرایی
آیا با توجه به بودجه و فناوری موجود می توان الزامات را اجرا کرد
قابلیت تایید
آیا می توان الزامات را بررسی کرد؟

Requirements validation techniques



✧ Requirements reviews

- Systematic **manual analysis** of the requirements.

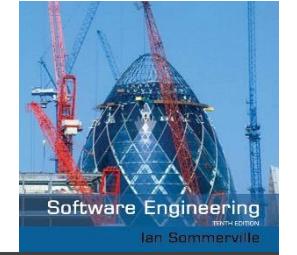
✧ Prototyping

- Using an **executable model of the system** to check requirements.

✧ Test-case generation

- Developing **tests** for requirements to check testability.

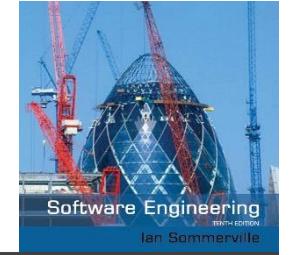
تکنیک های اعتبارسنجی نیازمندی ها
بررسی نیازمندی ها
.تجزیه و تحلیل دستی سیستماتیک نیازها
نمونه سازی
.استفاده از مدل اجرایی سیستم برای بررسی الزامات
تولید مورد آزمایشی
توسعه آزمایش هایی برای الزامات برای بررسی تست پذیری



Requirements reviews

- ✧ Regular reviews should be held while the requirements definition is being formulated.
- ✧ Both client and contractor staff should be involved in reviews.
- ✧ Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

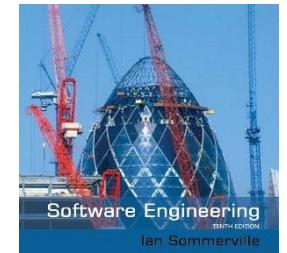
در حالی که تعریف الزامات در حال تدوین است، باید بررسی های منظم انجام شود. هم کارکنان کارفرما و هم کارکنان پیمانکار باید در بازبینی شرکت کنند. بررسی ها ممکن است رسمی (با مدارک تکمیل شده) یا غیر رسمی باشد. ارتباطات خوب بین توسعه دهنگان، مشتریان و کاربران می تواند مشکلات را در مراحل اولیه حل کند.



Requirements change

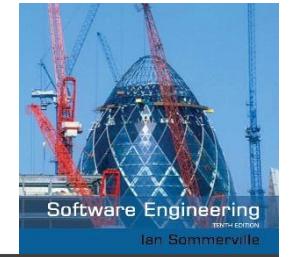
Changing requirements

ممکن است سختافزار جدیدی معرفی شود، ممکن است لازم باشد سیستم با سیستمهای دیگر ارتباط برقرار کند، اولویتهای تجاری ممکن است تغییر کند (در نتیجه تغییرات در پشتیبانی سیستم مورد نیاز است)، و قوانین و مقررات جدیدی ممکن است معرفی شوند که سیستم لزوماً باید از آنها پیروی کند.



- ❖ The business and technical environment of the system always changes after installation.
 - New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.
- ❖ The people who pay for a system and the users of that system are rarely the same people.
 - System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

محیط تجاری و فنی سیستم همیشه پس از نصب تغییر می کند.



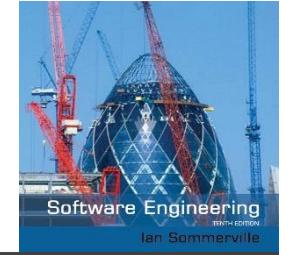
Changing requirements

- ✧ Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.
 - The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.

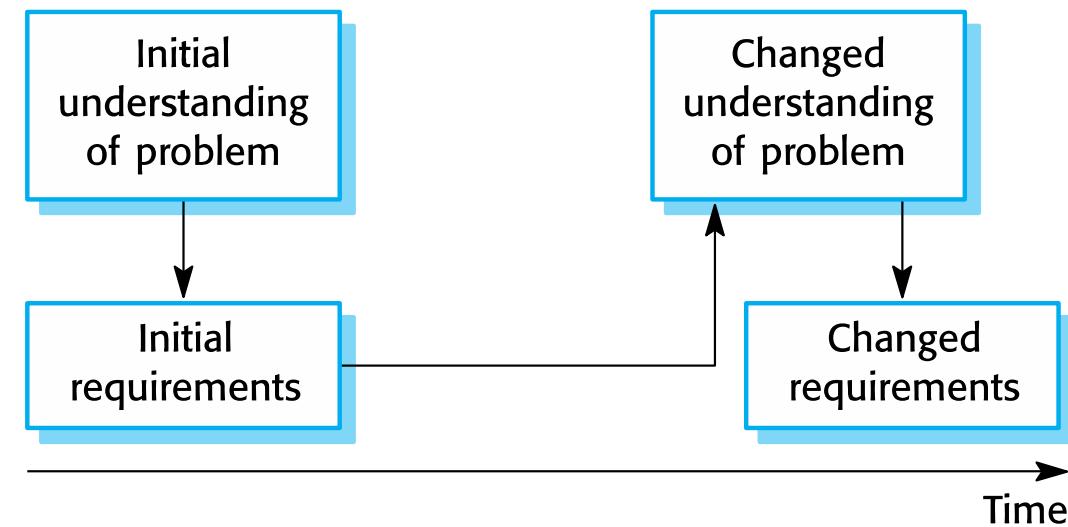
سیستمهای بزرگ معمولاً دارای جامعه کاربری متعدد و بسیاری از کاربران نیازمندیها و اولویت‌های متفاوتی دارند که ممکن است متناقض یا متناقض باشند.

الزامات سیستم نهایی به ناچار مصالحه ای بین آنهاست و با تجربه، اغلب کشف می شود که تعادل پشتیبانی ارائه شده به کاربران مختلف باید تغییر کند.

Requirements evolution

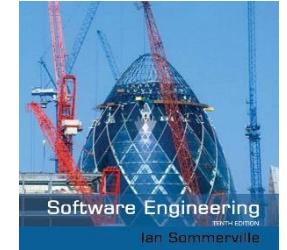


تکامل نیازمندی ها



Requirements management

مدیریت نیازمندی ها فرآیند مدیریت نیازمندی های در حال تغییر در طول فرآیند مهندسی نیازمندی ها و توسعه سیستم است. هنگامی که یک سیستم در حال توسعه و پس از استفاده از آن است، نیازهای جدید پدیدار می شوند.



- ✧ Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- ✧ New requirements emerge as a system is being developed and after it has gone into use.
- ✧ You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

شما باید نیازهای فردی را پیگیری کنید و پیوندهای بین نیازهای وابسته را حفظ کنید تا بتوانید تأثیر تغییرات نیازها را ارزیابی کنید. شما باید یک فرآیند رسمی برای ایجاد پیشنهادهای تغییر ایجاد کنید و آنها را به الزامات سیستم مرتبط کنید.

References

- **Dennis, Wixon, Tegarden, “System Analysis and Design, An Object Oriented Approach with UML”, 5th Edition, 2015.**
- **Summerville, “Software Engineering”, 10th Edition, 2014.**

Software Engineering I

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2021

Chapter 4

Functional Modeling(I)

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

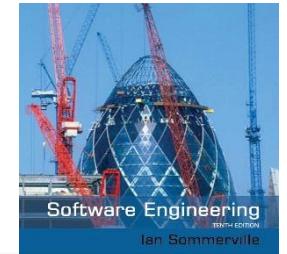
Introduction

تمام رویکردهای توسعه سیستم های شی گرا مبتنی بر موارد استفاده، معماری محور، و تکراری و افزایشی هستند.

- Use case نمایش نحوه تعامل یک سیستم تجاری با محیط خود است.
- Use case یک نمای کلی در سطح بالا از فرآیندهای تجاری در یک سیستم اطلاعات کسب و کار است.
- موارد استفاده کل پایه یک سیستم شی گرا را نشان می دهد.
- موارد استفاده می توانند سیستم فعلی (یعنی سیستم همانطور که هست) یا سیستم جدید در حال توسعه (یعنی سیستم آینده) را مستند کند.
- موارد استفاده همچنین پایه و اساس آزمایش و طراحی رابط کاربری را تشکیل می دهند.

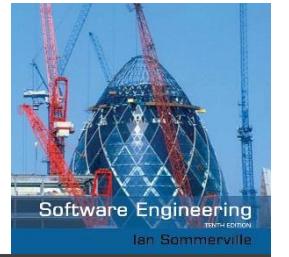
- All object-oriented systems development approaches are use-case driven, architecture-centric, and iterative and incremental.
- *Use case* is a formal way of representing the way a business system interacts with its environment.
- *Use case* is a high-level overview of the business processes in a business information system.
- *Use cases* represent the entire basis for an object-oriented system.
- *Use cases* can document the current system (i.e., as-is system) or the new system being developed (i.e., to-be system).
- *Use cases* also form the foundation for testing and user-interface design .

System modeling



- ✧ System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- ✧ System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- ✧ System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

مدلسازی سیستم فرآیند توسعه مدل‌های انتزاعی یک سیستم است که هر مدل دیدگاه یا دیدگاه متفاوتی از آن سیستم ارائه می‌کند. مدلسازی سیستم اکنون به معنای نمایش یک سیستم با استفاده از نوعی نماد گرافیکی است (UML) است، که اکنون تقریباً همیشه بر اساس نمادها در زبان مدلسازی واحد مدل‌سازی سیستم به تحلیلگر کمک می‌کند تا عملکرد سیستم را درک کند و از مدل‌ها برای برقراری ارتباط با مشتریان استفاده می‌شود.



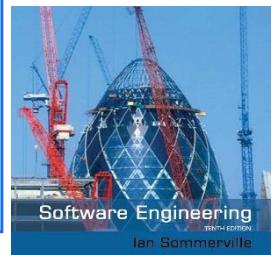
System perspectives

- ✧ An **external perspective**, where you **model the context or environment of the system**.
- ✧ A **structural perspective**, where you model the **organization of a system or the structure of the data** that is processed by the system.
- ✧ A **behavioral perspective**, where you model the **dynamic behavior** of the system and **how it responds to events**.

یک چشم انداز خارجی، که در آن زمینه یا محیط سیستم را مدل می کنید.
یک دیدگاه ساختاری، که در آن سازمان یک سیستم یا ساختار داده هایی را که توسط سیستم پردازش می شود مدل می کنید.
دیدگاه رفتاری، که در آن شما رفتار پویا سیستم و نحوه واکنش آن به رویدادها را مدل می کنید.

UML diagram types

از نمودارهای موردنی استفاده کنید که تعاملات بین یک سیستم و محیط آن را نشان می‌دهد.
نمودارهای فعالیت، که فعالیت‌های درگیر در یک فرآیند یا پردازش داده‌ها را نشان می‌دهد.
نمودارهای کلاس، که کلاس‌های شی در سیستم و ارتباط بین این کلاس‌ها را نشان می‌دهد.
نمودارهای توالی، که تعاملات بین بازیگران و سیستم و بین اجزای سیستم را نشان می‌دهد.
نمودارهای حالت، که نشان می‌دهد سیستم چگونه به رویدادهای داخلی و خارجی واکنش نشان می‌دهد.



- ❖ Use case diagrams, which show the interactions between a system and its environment.
- ❖ Activity diagrams, which show the activities involved in a process or in data processing .
- ❖ Class diagrams, which show the object classes in the system and the associations between these classes.
- ❖ Sequence diagrams, which show interactions between actors and the system and between system components.
- ❖ State diagrams, which show how the system reacts to internal and external events.

Introduction(Cnt'd)

- From an **architecture-centric perspective**, **use-case modeling** supports the creation of an **external or functional view** of a business process in that it shows **how the users** view the **process** rather than the **internal mechanisms** by which the process and supporting systems operate.

از دیدگاه معماری محور، مدلسازی مورد استفاده از ایجاد یک نمای خارجی یا عملکردی از یک فرآیند کسبوکار پشتیبانی میکند، زیرا نشان میدهد که کاربران چگونه فرآیند را میبینند. نه مکانیسمهای داخلی که فرآیند و سیستمهای پشتیبان عمل میکنند.

Introduction(Cnt'd)

- *Activity diagrams* are typically used to augment our understanding of the business processes and our use-case model.
- Technically, an activity diagram can be used for any type of process-modeling activity.

نمودارهای فعالیت معمولاً برای تقویت درک ما از فرآیندهای تجاری و مدل مورد استفاده ما استفاده می شود.
از نظر فنی، نمودار فعالیت را می توان برای هر نوع فعالیت مدلسازی فرآیند استفاده کرد.

نمودارهای فعالیت و موارد استفاده مدل‌های منطقی هستند – مدل‌هایی که فعالیتهای حوزه کسبوکار را بدون نشان دادن • نحوه انجام آنها توصیف می‌کنند

- مدل‌های منطقی گاهی اوقات به عنوان مدل‌های حوزه مشکل نامیده می‌شوند. خواندن یک نمودار مورد استفاده یا فعالیت، در اصل، نباید نشان دهد که یک فعالیت کامپیوتری یا دستی است
- این جزئیات فیزیکی در طول طراحی زمانی که مدل‌های منطقی به مدل‌های فیزیکی اصلاح می‌شوند، تعریف می‌شوند. این مدل‌ها اطلاعاتی را ارائه می‌دهند که برای ساختن سیستم مورد نیاز است
- ابتدا با تمرکز بر فعالیت‌های منطقی، تحلیلگران می‌توانند بر نحوه اجرای کسب و کار تمرکز کنند بدون اینکه حواسشان به جزئیات پیاده سازی منحرف شود

- Activity diagrams and use cases are *logical models*—models that describe the business domain's activities without suggesting how they are conducted.
- *Logical models* are sometimes referred to as *problem domain models*. Reading a use-case or activity diagram, in principle, should not indicate if an activity is computerized or manual.
- These physical details are defined during design when the logical models are refined into *physical models*. These models provide information that is needed to ultimately build the system.
- By focusing on logical activities first, analysts can focus on how the business should run without being distracted with implementation details.

Use-case Diagram

- Employ the use-case diagram to better understand the functionality of the system at a very high level.
- Because a use-case diagram provides a simple, straightforward way of communicating to the users exactly what the system will do, a use-case diagram is drawn when gathering and defining requirements for the system.
- Use-case diagram can encourage the users to provide additional high-level requirements.
- A use-case diagram illustrates in a very simple way the main functions of the system and the different kinds of users that will interact with it

از نمودار استفاده برای درک بهتر عملکرد سیستم در سطح بسیار بالا استفاده کنید.

- از آنجایی که نمودار مورد استفاده راهی ساده و سرراست برای برقراری ارتباط با کاربران دقیقاً آنچه که سیستم انجام خواهد داد، ارائه می‌کند.
- هنگام جمعاًوری و تعریف الزامات برای سیستم، نمودار مورد استفاده ترسیم می‌شود.
- نمودار مورد استفاده می‌تواند کاربران را ترغیب کند تا نیازهای سطح بالا را فراهم کنند.
- یک نمودار مورد استفاده به روشی بسیار ساده عملکردهای اصلی سیستم و انواع مختلف کاربرانی که با آن تعامل خواهند داشت را نشان می‌دهد.

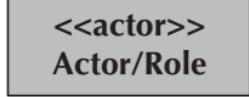
Let's start

- For identifying **use cases**, Jacobson et al. (1992) recommend that you ask the following questions:
 - What are the **main tasks performed by each actor**?
 - Will the **actor read** or **update** any information in the system?
 - Will the **actor have to inform** the system about **changes outside the system**?
 - Does the actor have to **be informed of unexpected changes**?

وظایف اصلی هر بازیگر چیست؟ •

- آیا بازیگر اطلاعات موجود در سیستم را می خواند یا به روز می کند؟ •
- آیا بازیگر باید تغییرات خارج از سیستم را به اطلاع سیستم برساند؟ •
- آیا بازیگر باید از تغییرات غیرمنتظره مطلع شود؟ •

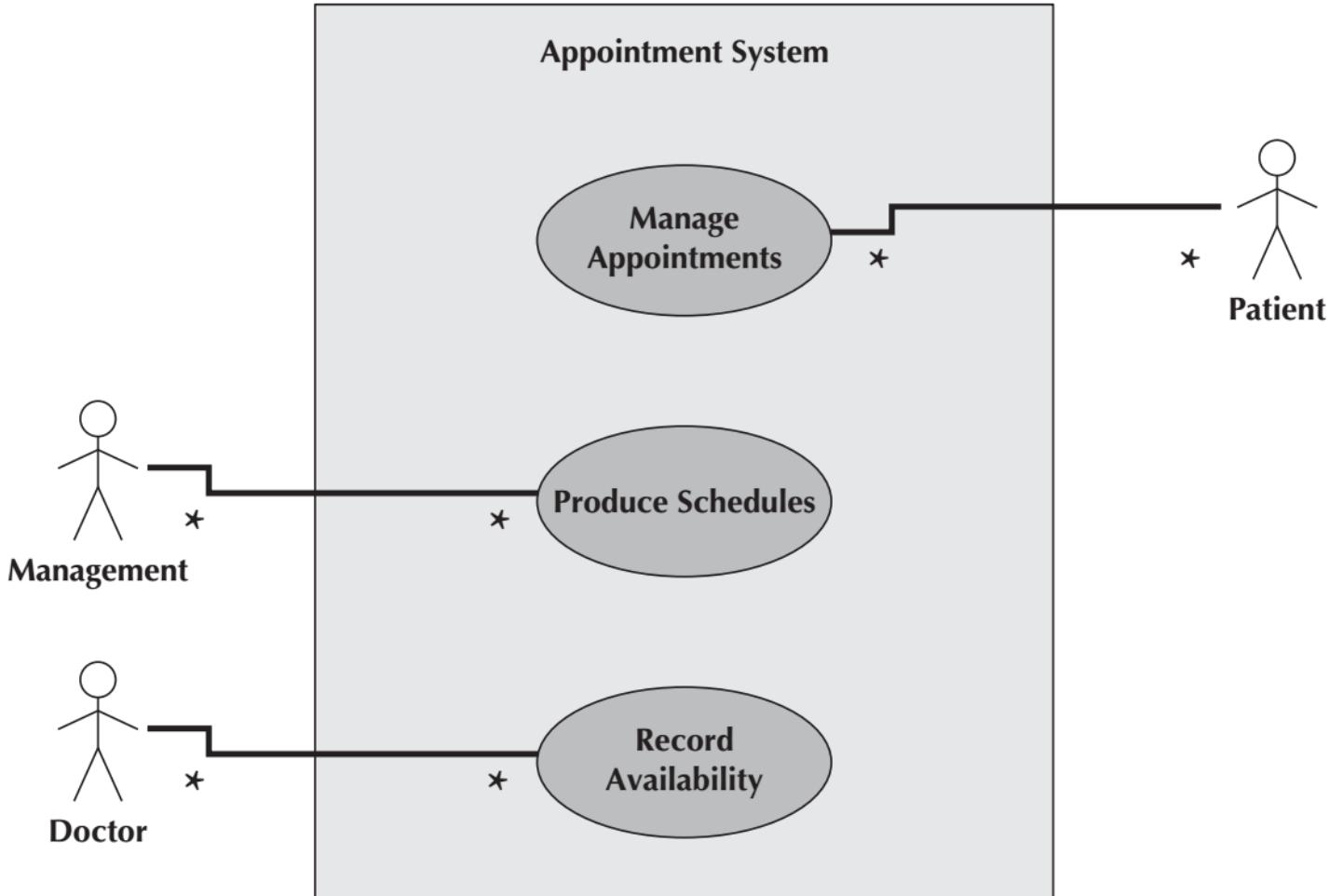
Elements of Use-Case Diagrams(I)

<p>An actor:</p> <ul style="list-style-type: none">■ Is a person or system that derives benefit from and is external to the subject.■ Is depicted as either a stick figure (default) or, if a nonhuman actor is involved, a rectangle with <<actor>> in it (alternative).■ Is labeled with its role.■ Can be associated with other actors using a specialization/superclass association, denoted by an arrow with a hollow arrowhead.■ Is placed outside the subject boundary.	 <p>Actor/Role</p>  <p><<actor>> Actor/Role</p>
<p>A use case:</p> <ul style="list-style-type: none">■ Represents a major piece of system functionality.■ Can extend another use case.■ Can include another use case.■ Is placed inside the system boundary.■ Is labeled with a descriptive verb–noun phrase.	 <p>Use Case</p>
<p>A subject boundary:</p> <ul style="list-style-type: none">■ Includes the name of the subject inside or on top.■ Represents the scope of the subject, e.g., a system or an individual business process.	 <p>Subject</p>

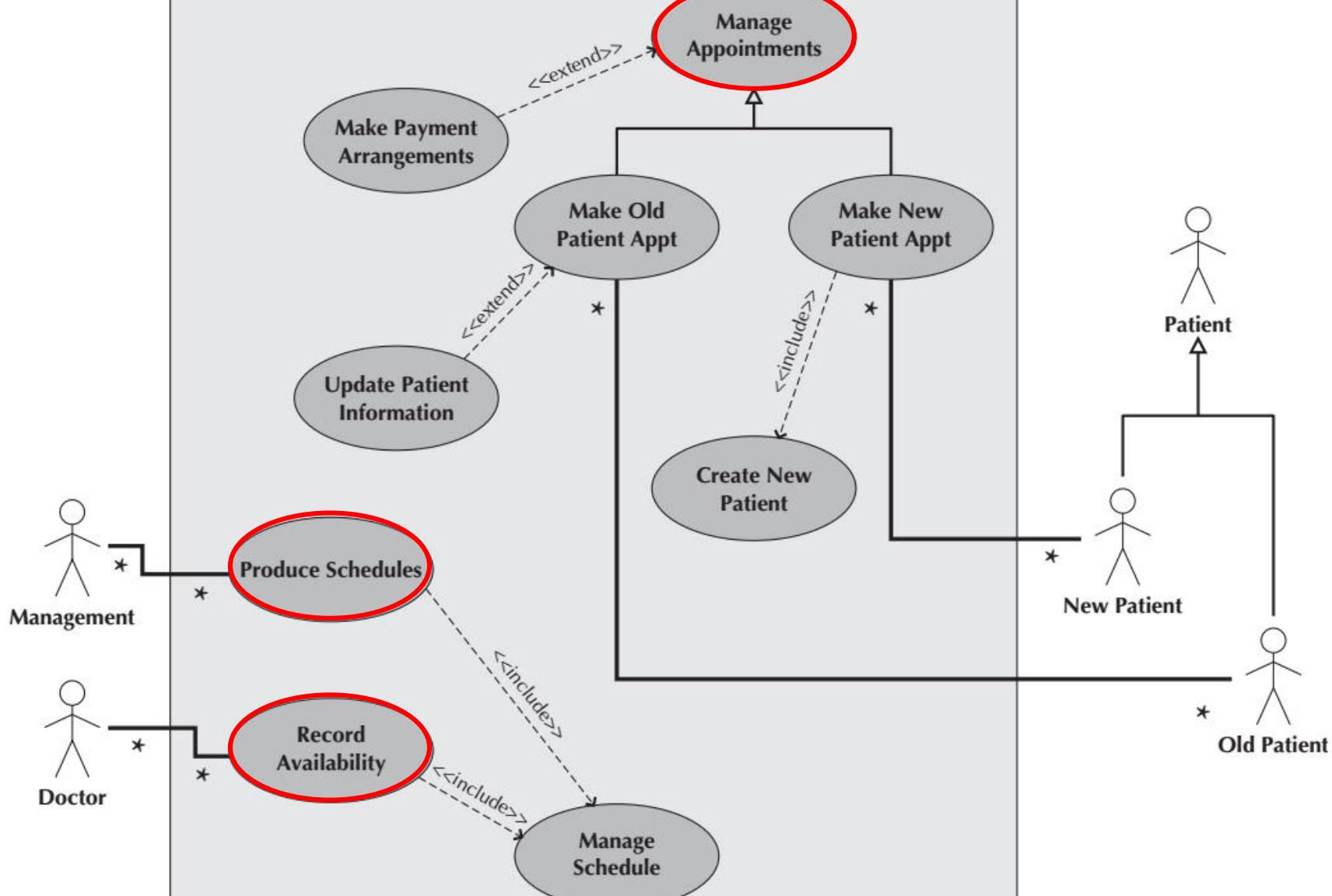
Elements of Use-Case Diagrams(II)

<p>An association relationship:</p> <ul style="list-style-type: none">Links an actor with the use case(s) with which it interacts.	
<p>An include relationship:</p> <ul style="list-style-type: none">Represents the inclusion of the functionality of one use case within another.Has an arrow drawn from the base use case to the used use case.	
<p>An extend relationship:</p> <ul style="list-style-type: none">Represents the extension of the use case to include optional behavior.Has an arrow drawn from the extension use case to the base use case.	
<p>A generalization relationship:</p> <ul style="list-style-type: none">Represents a specialized use case to a more generalized one.Has an arrow drawn from the specialized use case to the base use case.	

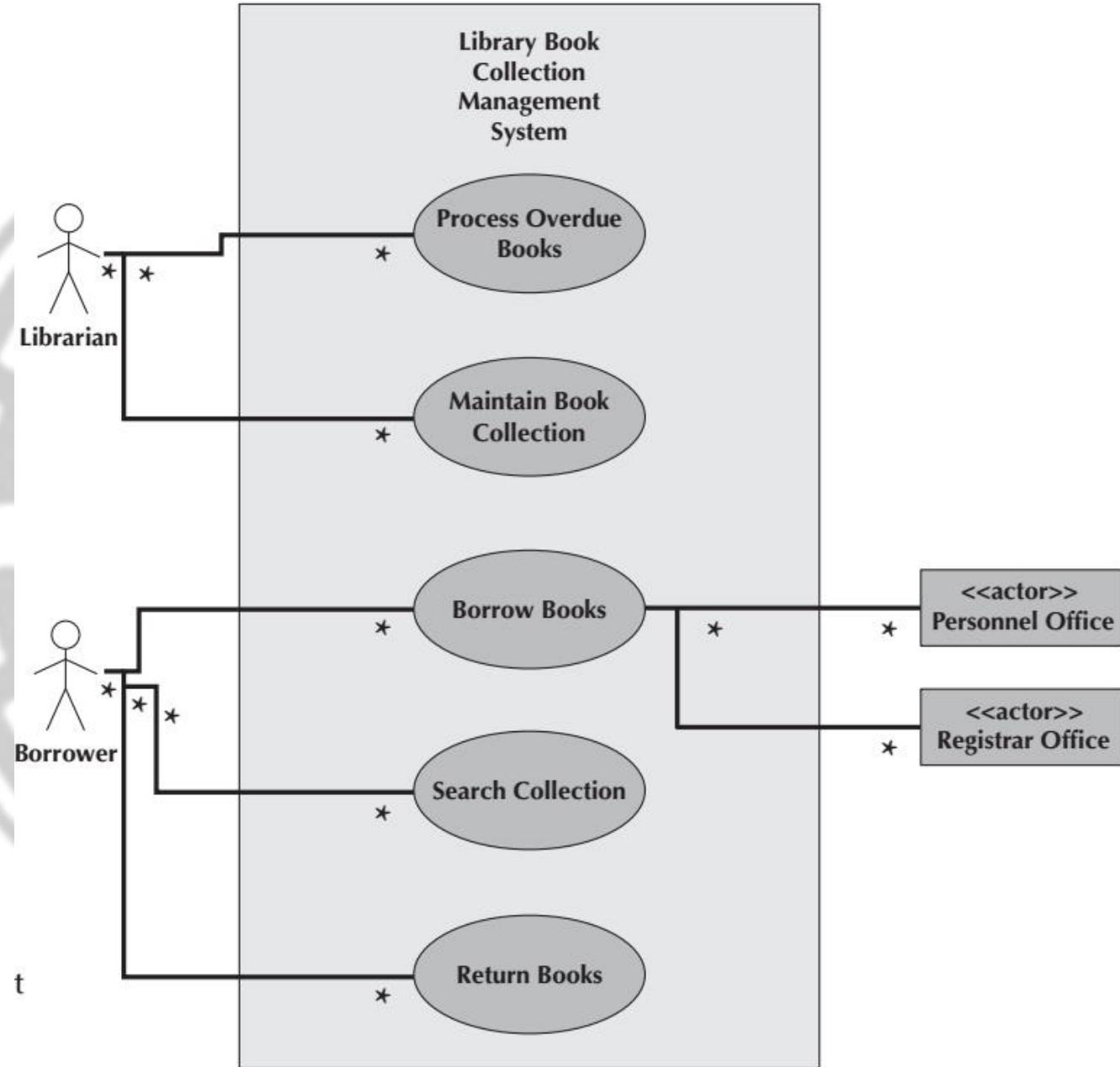
First Example



Appointment System



Second example



What should you do for your project?

1. Create use-case diagram, level 0.

We will work in the lab.

Reference

- **Dennis, Wixon, Tegarden, “System Analysis and Design, An Object Oriented Approach with UML”, 5th Edition, 2015.**
- **Valacich, J. S., J. F. George, “Modern systems analysis and design”, 8th Edition, 2017.**

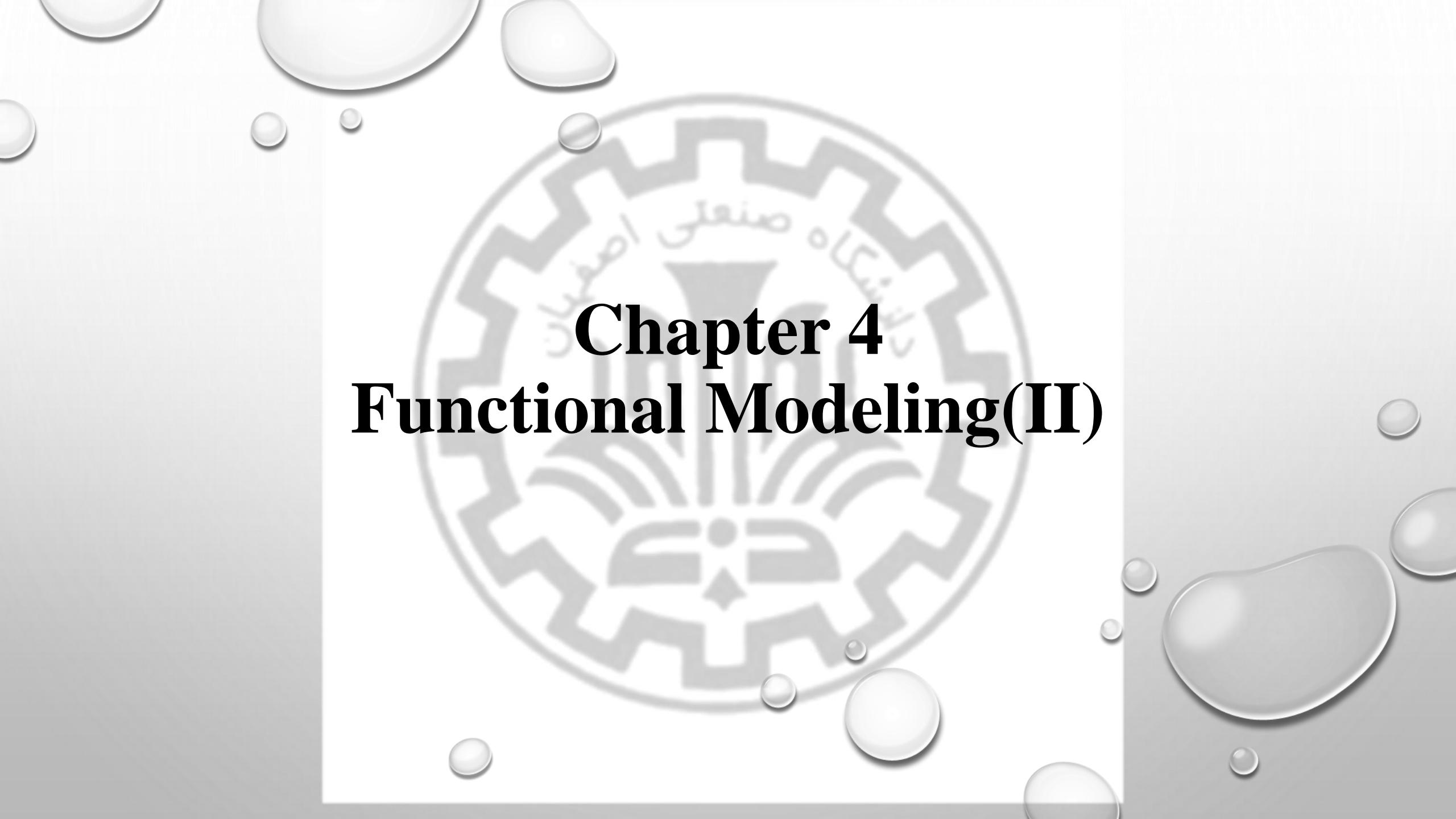
Software Engineering I

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2021



Chapter 4

Functional Modeling(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

Business Process Modelling

- Business process models describe the different activities that, when combined, support a business process.
- From an object-oriented perspective, these processes cut across multiple roles.

Activity diagrams are used to model the behavior in a business process.

- مدل‌های فرآیند کسبوکار، فعالیتهای مختلفی را توصیف می‌کنند که در صورت ترکیب، از یک فرآیند تجاری پشتیبانی می‌کنند.
- از دیدگاه شی گرا، این فرآیندها نقش‌های متعددی را برش می‌دهند. نمودارهای فعالیت برای مدل‌سازی رفتار در یک فرآیند تجاری استفاده می‌شود.

Activity diagram

- It can be used to model everything from a **high-level business workflow**.
- Can model a business process **independent** of any object implementation.
- It can be used at a **high level** as well as at a **low level** of abstraction.
- At a conceptual level, an **activity is a task** that needs to be done, whether by a human or a computer.
- At an **implementation** level, an **activity is a method or a class**.

می توان از آن برای مدل سازی همه چیز از یک گردش کار تجاری سطح بالا استفاده کرد •

می تواند فرآیند کسب و کار را مستقل از اجرای هر شی مدل کند •

می توان از آن در سطح بالا و همچنین در سطح پایین انتزاع استفاده کرد •

در سطح مفهومی، یک فعالیت وظیفه ای است که باید انجام شود، چه توسط یک انسان یا یک کامپیوتر •

در سطح پیاده سازی، یک اکتیویتی یک متاد یا یک کلاس است •

When to use an activity diagram

- It should be used **only when it adds value to the project.**
- Ask the following question: **Does it add value or is it redundant?**
- An activity diagram can **be used to accomplish the following tasks.**
 1. **Depict the flow of control** from activity to activity.
 2. Help in **use case analysis** to understand **what actions** need to take place
 3. Help in **identifying extensions** in a use case.
 4. Model **work flow** and **business processes**.
 5. Model the sequential and concurrent steps in a **computation process**.

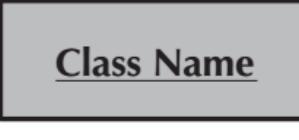
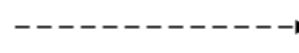
فقط زمانی باید از آن استفاده کرد که به پروژه ارزش اضافه کند.

سوال زیر را بپرسید: آیا ارزش افزوده دارد یا اضافی است؟
برای انجام وظایف زیر می توان از نمودار فعالیت استفاده کرد.

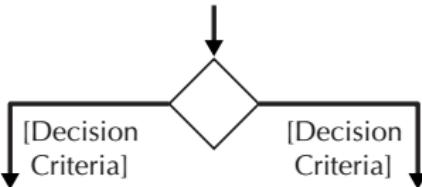
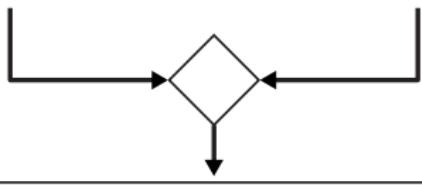
جریان کنترل را از فعالیتی به فعالیت دیگر به تصویر بکشید.
کمک به تجزیه و تحلیل موارد استفاده برای درک اینکه چه اقداماتی باید انجام شوند
کمک به شناسایی برنامه های افزودنی در یک مورد استفاده.

مدل سازی جریان کار و فرآیندهای کسب و کار.
مدلسازی مراحل متوالی و همزمان در یک فرآیند محاسباتی.

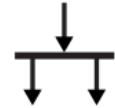
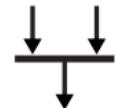
Elements of an Activity Diagram(I)

An action: <ul style="list-style-type: none">■ Is a simple, nondecomposable piece of behavior.■ Is labeled by its name.	
An activity: <ul style="list-style-type: none">■ Is used to represent a set of actions.■ Is labeled by its name.	
An object node: <ul style="list-style-type: none">■ Is used to represent an object that is connected to a set of object flows.■ Is labeled by its class name.	
A control flow: <ul style="list-style-type: none">■ Shows the sequence of execution.	
An object flow: <ul style="list-style-type: none">■ Shows the flow of an object from one activity (or action) to another activity (or action).	

Elements of an Activity Diagram(II)

An initial node: <ul style="list-style-type: none">Portrays the beginning of a set of actions or activities.	
A final-activity node: <ul style="list-style-type: none">Is used to stop all control flows and object flows in an activity (or action).	
A final-flow node: <ul style="list-style-type: none">Is used to stop a specific control flow or object flow.	
A decision node: <ul style="list-style-type: none">Is used to represent a test condition to ensure that the control flow or object flow only goes down one path.Is labeled with the decision criteria to continue down the specific path.	
A merge node: <ul style="list-style-type: none">Is used to bring back together different decision paths that were created using a decision node.	

Elements of an Activity Diagram(III)

A fork node: Is used to split behavior into a set of parallel or concurrent flows of activities (or actions)	
A join node: Is used to bring back together a set of parallel or concurrent flows of activities (or actions)	
A swimlane: Is used to break up an activity diagram into rows and columns to assign the individual activities (or actions) to the individuals or objects that are responsible for executing the activity (or action) Is labeled with the name of the individual or object responsible	

می تواند رفتار دستی یا کامپیوتري را نشان دهد.

در نمودار فعالیت به صورت یک مستطیل گرد به تصویر کشیده شده است.

آنها باید نامی داشته باشند که با یک فعل شروع و با یک اسم ختم شود (به عنوان مثال، اطلاعات بیمار را دریافت کنید).

نام ها باید کوتاه باشند، در عین حال حاوی اطلاعات کافی باشند تا خواننده به راحتی متوجه شود که دقیقاً چه کاری انجام می دهد.

نتها تفاوت بین یک عمل و یک فعالیت این است که یک فعالیت را می توان بیشتر به مجموعه ای از فعالیت ها و/یا کنش ها تجزیه کرد، در

حالی که یک کنش نشان دهنده یک قطعه ساده غیرقابل تجزیه از رفتار کلی است که مدل سازی می شود.

در بیشتر موارد، هر فعالیت با یک مورد استفاده همراه است.

Actions and Activities

- Can represent **manual or computerized behavior**.
- Depicted in an activity diagram as a **rounded rectangle**.
- They should have a name that **begins with a verb** and **ends with a noun** (e.g., Get Patient Information)
- Names should be **short**, yet contain **enough information** so that the reader can easily understand exactly what they do.
- The **only difference** between an **action** and an **activity** is that an activity can be **decomposed** further into a set of activities and/or actions, whereas an **action** represents a simple non-decomposable piece of the overall behavior being modeled.
- In most cases, **each activity** is associated with a use case.

Control Flows

- *Control flows* model the paths of execution through a business process.
- A control flow is portrayed as a solid line with an arrowhead on it showing the direction of flow.
- Control flows can be attached only to actions or activities.

جريان های کنترلی مسیرهای اجرا را از طریق یک فرآیند تجاری مدل می کنند.

- یک جریان کنترلی به صورت یک خط ثابت با نوک پیکانی که جهت جریان را نشان می دهد به تصویر کشیده می شود.
- جریان های کنترلی را می توان فقط به اقدامات یا فعالیت ها متصل کرد.

Initial node

- Portrays the beginning of a set of actions or activities.
- Is shown as a small filled-in circle.

آغاز مجموعه ای از اقدامات یا فعالیت ها را
به تصویر می کشد.
به صورت یک دایره کوچک پر شده نشان داده
شده است.

final-activity node

- Is used to stop the process being modeled.
- Any time a final-activity node is reached, all actions and activities are ended immediately, regardless of whether they are completed.
- Is represented as a circle surrounding a small, filled-in circle.

برای توقف فرآیند در حال مدل سازی استفاده می شود .
هر زمان که به یک گره فعالیت نهایی رسید، تمام اقدامات و فعالیت ها بدون .
در نظر گرفتن اینکه کامل شده باشند، فوراً پایان می یابند
به صورت دایره ای نشان داده می شود که یک دایره کوچک و پر شده را .
احاطه کرده است.

Final-flow node

- Is similar to a final-activity node, except that it stops a specific path of execution through the business process but allows the other concurrent or parallel paths to continue.
- Is shown as a small circle with an X in it.

شبیه یک گره فعالیت نهایی است، با این تفاوت که یک مسیر خاص از اجرا را از طریق فرآیند کسب و کار متوقف می کند، اما اجازه می دهد تا مسیرهای همزمان یا موازی دیگر ادامه پیدا کنند در آن نشان داده شده است X به صورت یک دایره کوچک با •

Decision node

- Is used to represent the **actual test condition** that determines which of the paths exiting the decision node is to be traversed.
- In this case, each exiting path must **be labeled with a guard condition**.
- A *guard condition* represents the **value of the test** for that particular path to be executed.

برای نشان دادن شرایط آزمایش واقعی استفاده می شود که تعیین می کند کدام یک از مسیرهای خروجی از گره تصمیم باید طی شود.
در این حالت، هر مسیر خروجی باید با یک شرط محافظت برچسب گذاری شود •
یک شرط نگهبان نشان دهنده مقدار آزمایش برای آن مسیر خاص است که باید اجرا شود.

Merge node

- Is used to bring **back together** multiple mutually **exclusive paths** that have been split based on earlier.
- However, sometimes, **for clarity**, it is better not to use a merge node.

برای گردآوری چندین مسیر متقابل منحصر به فرد که بر اساس قبل تقسیم شده
آن استفاده می شود.
با این حال، گاهی اوقات، برای وضوح، بهتر است از گره ادغام استفاده نکنید.

Fork and Join nodes

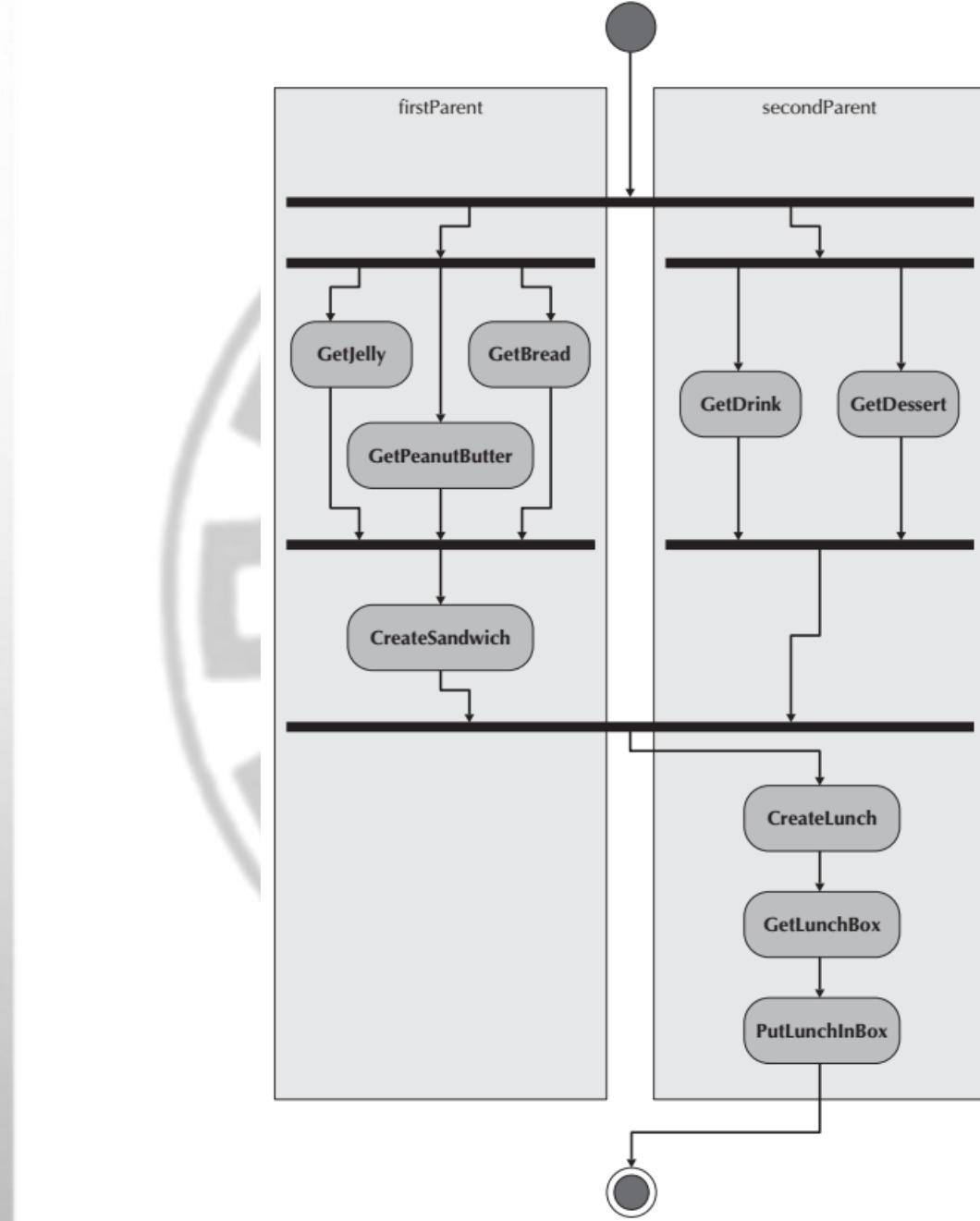
- The fork and join nodes allow parallel and concurrent processes to be modeled .
- The *fork node* is used to split the behavior of the business process into multiple parallel or concurrent flows. Unlike the decision node, the paths are not mutually exclusive.
- The *join node* simply brings back together the separate parallel or concurrent flows in the business process into a single flow.

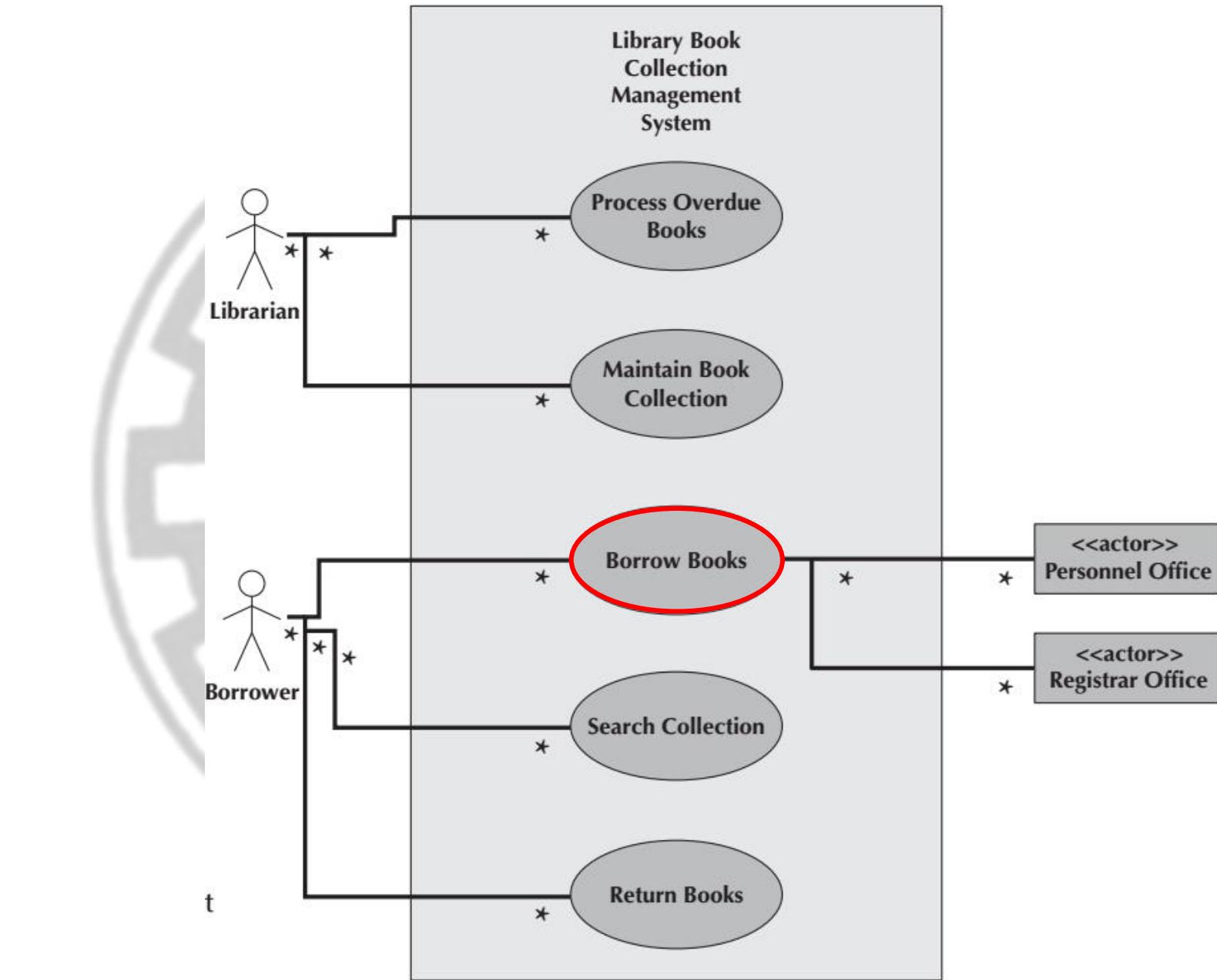
گره های فورک و اتصال به فرآیندهای موازی و همزمان اجازه مدل سازی می دهد.
گره فورک برای تقسیم رفتار فرآیند کسب و کار به چند جریان موازی یا همزمان استفاده می شود .
برخلاف گره تصمیم، مسیرها متقابل نیستند
گره اتصال به سادگی جریان های موازی یا همزمان جداگانه در فرآیند کسب و کار را به یک جریان .
واحد برمی گرداند.

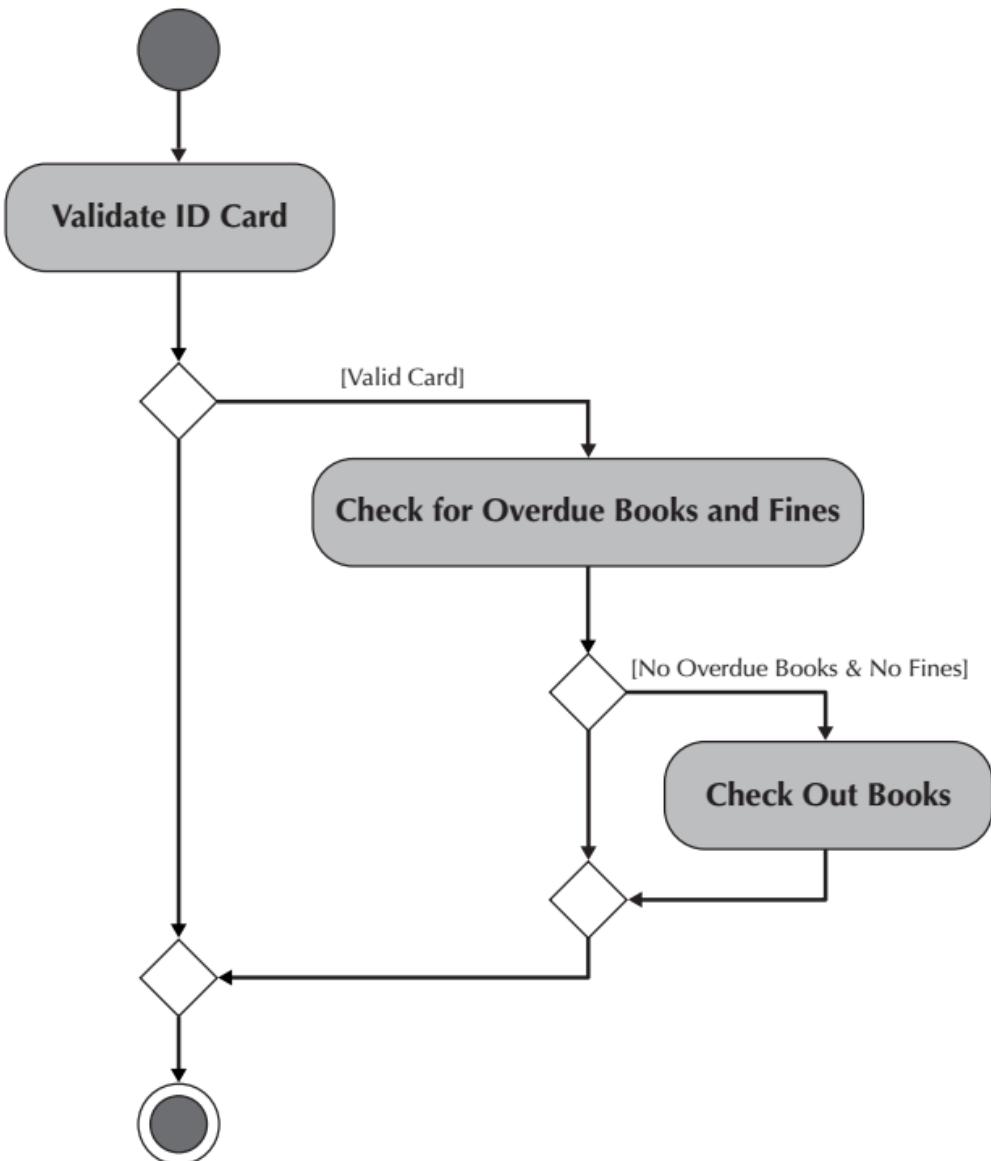
Swimlanes

- However, there are times when it helps to break up an activity diagram in such a way that it can be used to assign responsibility to objects or individuals who would actually perform the activity. This is especially useful when modeling a business workflow and is accomplished through the use of *swimlanes*.

با این حال، موقعي وجود دارد که به شکستن نمودار فعالیت کمک می کند به گونه ای که بتوان از آن برای واگذاری مسئولیت به اشیا یا افرادی که واقعاً فعالیت را انجام می دهند استفاده کرد. این امر مخصوصاً هنگام مدلسازی یک گردش کار تجاری مفید است و از انجام میشود **swimlanes** طریق استفاده از







Object nodes

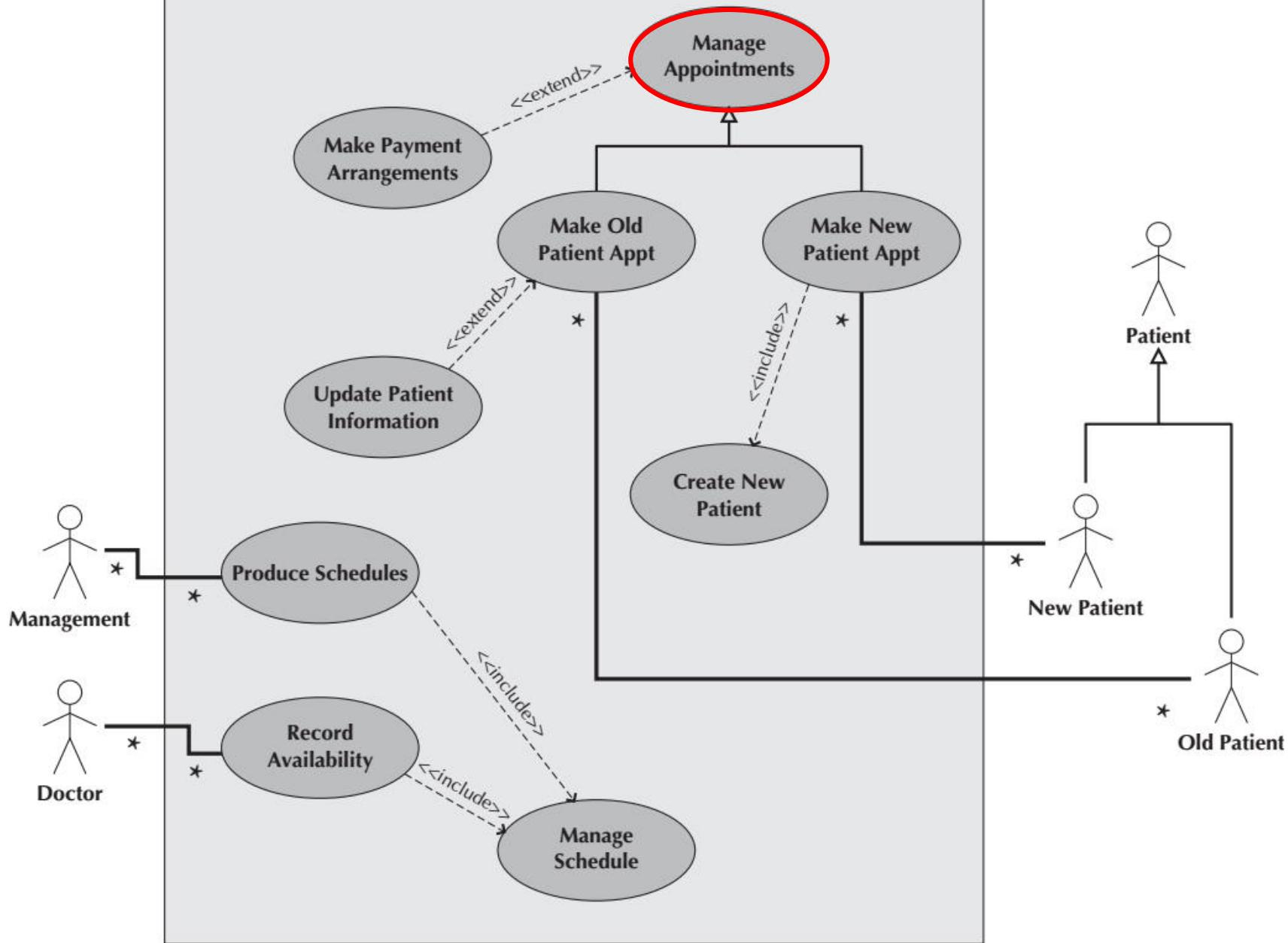
- Activities and actions typically modify or transform objects. *Object nodes* model these objects in an activity diagram.
- The name of the class of the object is written inside the rectangle.
- Essentially, object nodes represent the flow of information from one activity to another activity.

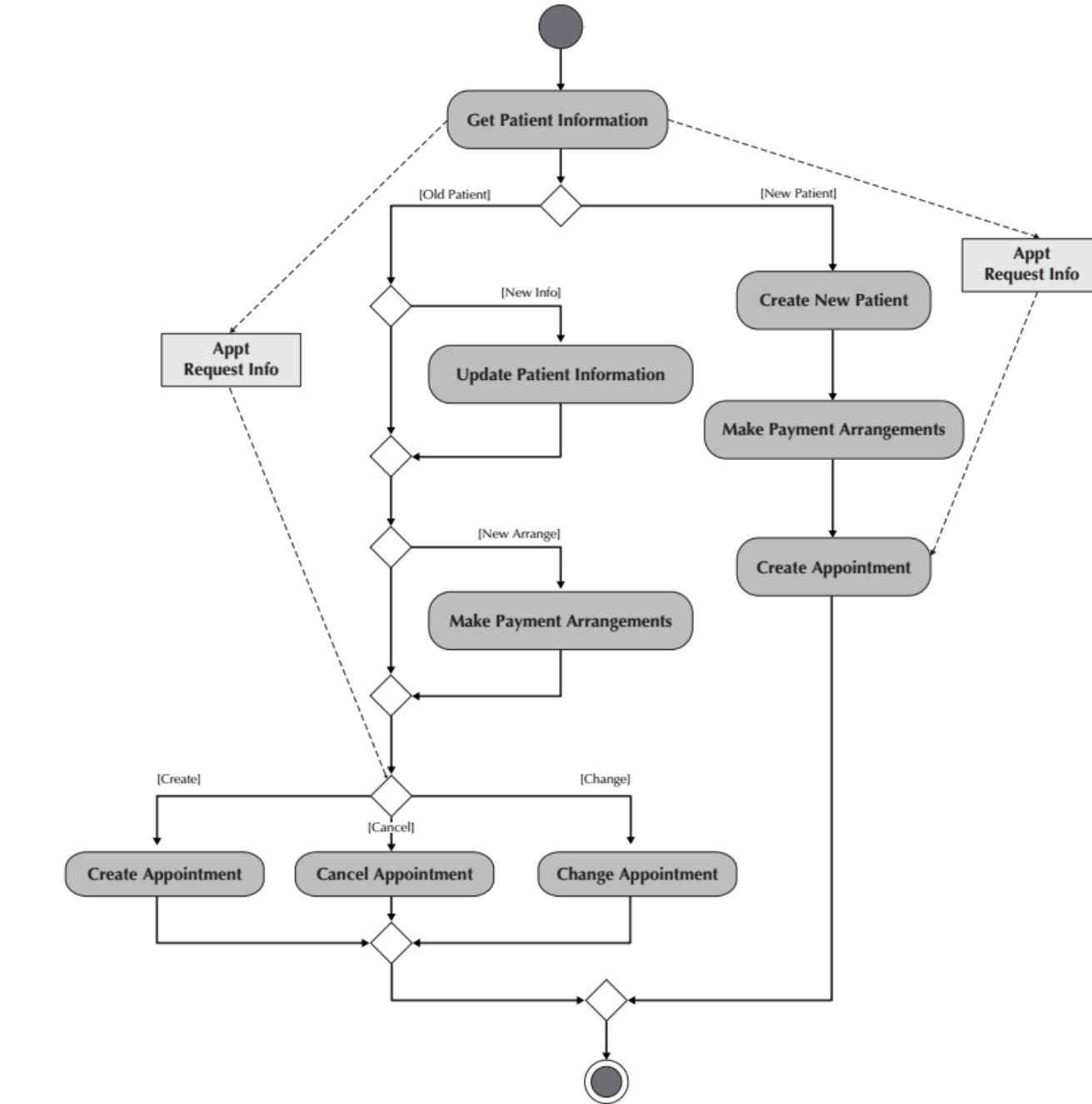
فعالیت ها و اعمال معمولاً اشیاء را تغییر می دهند یا تغییر می دهند.
گره های شیء این اشیاء را در یک نمودار فعالیت مدل می کنند
نام کلاس شیء داخل مستطیل نوشته می شود •
اساساً، گره های شی جریان اطلاعات را از یک فعالیت به فعالیت •
دیگر نشان می دهند

Object Flows

- *Object flows* model the flow of objects through a business process.
- Because activities and actions modify or transform objects, object flows are necessary to show the actual objects that flow into and out of the actions or activities.
- An object flow is depicted as a dashed line with an arrowhead on it showing the direction of flow.
- An individual object flow must be attached to an action or activity on one end and an object node on the other end.

Appointment System



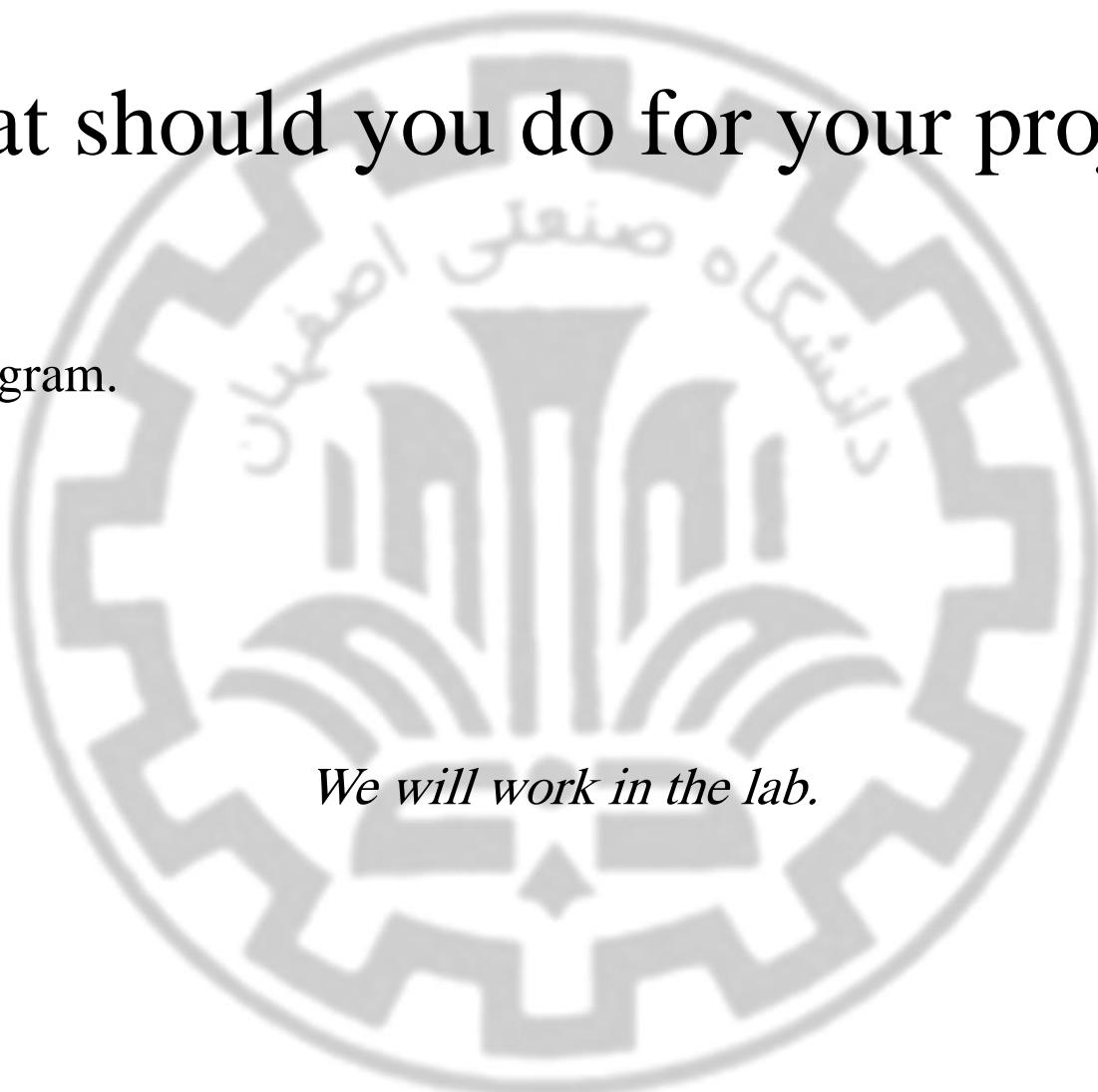


Control Nodes

- Initial,
- Final-activity,
- Final-flow,
- Decision,
- Merge,
- Fork,
- Join

What should you do for your project?

1. Create Activity diagram.



We will work in the lab.

Reference

- **Dennis, Wixon, Tegarden, “System Analysis and Design, An Object Oriented Approach with UML”, 5th Edition, 2015.**
- **Valacich, J. S., J. F. George, “Modern systems analysis and design”, 8th Edition, 2017.**

Software Engineering I

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2021

Chapter 4

Functional Modeling(III)

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

Introduction

- Use-case diagrams provided a bird's-eye view of the basic functionality of the business processes contained in the evolving system.
- Activity diagrams, in a sense, open up the black box of each business process by providing a more-detailed graphical view of the underlying activities that support each business process.
- Use-case descriptions provide a means to more fully document the different aspects of each individual use case.

نمودارهای مورد استفاده یک نمای چشم پرندۀ از عملکرد اساسی فرآیندهای تجاری موجود در سیستم در حال تکامل ارائه می‌دهد.

نمودارهای فعالیت، به یک معنا، جعبه سیاه هر فرآیند کسب و کار را با ارائه یک نمای گرافیکی با جزئیات بیشتر از فعالیت‌های اساسی که از هر فرآیند تجاری پشتیبانی می‌کنند، باز می‌کند.

توضیحات مورد استفاده وسیله‌ای برای مستندسازی کاملتر جنبه‌های مختلف هر مورد استفاده فردی فراهم می‌کند.

The use-case descriptions are based on the identified requirements, use-case diagram, and the activity diagram of the business processes.

Use-case descriptions contain **all the information needed** to document the functionality of the business processes.

توضیحات مورد استفاده بر اساس الزامات شناسایی شده، نمودار مورد استفاده و نمودار فعالیت فرآیندهای تجاری است. توضیحات مورد استفاده شامل تمام اطلاعات مورد نیاز برای مستندسازی عملکرد فرآیندهای تجاری است.

Introduction

- Use cases are the primary drivers for all the UML diagramming techniques.
- A **use case** communicates at a **high level** what the system needs to do, and all the UML diagramming techniques build on this by presenting the use-case functionality in a different way for a different purpose.
- Use cases are the building blocks by which the system is designed and built.

هستند **UML** موارد استفاده، محرك های اولیه برای تمام تکنیک های نمودار.
مورد استفاده در سطح بالایی آنچه را که سیستم باید انجام دهد ارتباط برقرار می کند، و تمام تکنیک های نمودار.
بر این اساس با ارائه عملکرد مورد استفاده به روشی متفاوت برای هدفی متفاوت ساخته می شوند **UML**
موارد استفاده بلوک های ساختمانی هستند که سیستم توسط آنها طراحی و ساخته می شود.

Introduction

- Use cases capture the typical **interaction of the system with the system's users** (end users and other systems).
- These interactions represent the **external, or functional view** of the system from the perspective of the user.
- Each use case describes **one and only one function** in which users interact with the system.
- Although a use case may contain several paths that a user can take while interacting with the system, **each possible execution path through the use case** is referred to as a *scenario*.

• موارد استفاده تعامل معمولی سیستم با کاربران سیستم (کاربران نهایی و سایر سیستم‌ها) را نشان می‌دهد.

• این فعل و انفعالات نمای خارجی یا عملکردی سیستم را از دیدگاه کاربر نشان می‌دهد.

• هر مورد استفاده یک و تنها یک عملکرد را توصیف می‌کند که در آن کاربران با سیستم تعامل دارند.

• اگرچه یک مورد استفاده ممکن است شامل چندین مسیر باشد که کاربر می‌تواند در هین تعامل با سیستم طی کند، اما هر مسیر.

• اجرای ممکن از طریق مورد استفاده به عنوان یک سناریو نامیده می‌شود.

هنگام ایجاد توضیحات مورد استفاده، تیم پروژه باید از نزدیک با کاربران همکاری کند تا الزامات عملکردی را به طور کامل مستند کند.

- سازماندهی نیازمندیهای عملکردی و مستندسازی آنها در یک توصیف موردنی فرآیندی نسبتاً ساده است، اما برای اطمینان از کامل بودن توصیفات به اندازه کافی برای استفاده در مدلسازی ساختاری و رفتاری، تمرین قابل توجهی نیاز است.
 - بهترین مکان برای شروع مرور نمودارهای مورد استفاده و فعالیت است
 - این امکان وجود دارد که چندین کاربر نقش یکسانی را ایفا کنند. بنابراین، موارد استفاده باید با نقشهایی که کاربران بازی میکنند مرتبط باشد نه با خود کاربران.

- When creating use-case descriptions, the project team must work closely with the users to fully document the functional requirements.
- Organizing the functional requirements and documenting them in a use-case description are a relatively simple process, but it takes considerable practice to ensure that the descriptions are complete enough to use in structural and behavioral modeling.
- The best place to begin is to review the use-case and activity diagrams.
- It is possible that multiple users will play the same role. Therefore, use cases should be associated with the roles played by the users and not with the users themselves.

Types of Use Cases

- Classify a use case based on the purpose of the use case and the amount of information that the use case contains: **overview** versus **detail** and **essential** versus **real**.

طبقه بندی یک مورد استفاده بر اساس هدف مورد استفاده و مقدار اطلاعاتی که مورد استفاده دارد: نمای کلی در مقابل جزئیات و ضروری در مقابل واقعی

Overview use case vs. detail use case

- *Over view use case* is used to enable the analyst and user to agree on a high-level overview of the requirements.
 - Typically, overview use cases are created very early in the process of understanding the system requirements, and they document only basic information about the use case.
 - These can easily be created immediately after the creation of the use-case diagram.
- A *detail use case* typically documents, as far as possible, all the information needed for the use case.

استفاده می شود تا تحلیلگر و کاربر را قادر سازد تا بر روی یک نمای کلی سطح بالا از الزامات توافق کنند Over view مورد استفاده از معمولاً موارد استفاده اجمالی در مراحل اولیه در رک الزامات سیستم ایجاد می شوند و آنها فقط اطلاعات اولیه در مورد موارد استفاده را مستند می کنند.

اینها را می توان به راحتی بلافاصله پس از ایجاد نمودار مورد استفاده ایجاد کرد .
یک مورد استفاده از جزئیات معمولاً تا آنجا که ممکن است، تمام اطلاعات مورد نیاز برای مورد استفاده را مستند می کند .
اینها می توانند بر اساس فعالیت ها و جریان های کنترلی موجود در نمودارهای فعالیت باشند .

Essential use case vs. real use case

- An **essential use case** is one that describes only the **minimum essential issues** necessary to understand the required functionality.
- A **real use case** goes farther and describes a specific set of steps.
- The primary difference is that **essential use cases** are **implementation independent**, whereas **real use cases** are **detailed descriptions** of how to use the system once it is implemented.
- Thus, real use cases tend to be used **only in the design, implementation, and testing**.

یک مورد استفاده ضروری موردنی است که فقط حداقل مسائل ضروری را که برای درک عملکرد مورد نیاز ضروری است، توصیف می کند •

یک مورد استفاده واقعی فراتر می رود و مجموعه خاصی از مراحل را توصیف می کند •

تفاوت اصلی این است که موارد استفاده ضروری مستقل از پیاده سازی هستند، در حالی که موارد استفاده واقعی، شرح مفصلی از نحوه استفاده از سیستم پس از پیاده سازی است •

بنابراین، موارد استفاده واقعی تنها در طراحی، اجرا و آزمایش مورد استفاده قرار می گیرند

Elements of a Use-Case Description

- A use-case description contains all the information needed to build the structural and behavioral diagrams that follow, but it expresses the information in a less-formal way that is usually simpler for users to understand.
- A use-case description has three basic parts: **overview information**, **relationships**, and the **flow of events**.

توصیف مورد استفاده شامل تمام اطلاعات مورد نیاز برای ساختن نمودارهای ساختاری و رفتاری است که در ادامه می‌آید، اما اطلاعات را به رو شی کمتر رسمی که معمولاً برای کاربران قابل درک است، بیان می‌کند.

توصیف مورد استفاده دارای سه بخش اساسی است: اطلاعات کلی، روابط، و جریان رویدادها.

overview

Relationships

Flow of events

Use Case Name: Make Old Patient Appt	ID: 2	Importance Level: Low
Primary Actor: Old Patient	Use Case Type: Detail, Essential	
Stakeholders and Interests: Old Patient – wants to make, change, or cancel an appointment Doctor – wants to ensure patient's needs are met in a timely manner		
Brief Description: This use case describes how we make an appointment as well as changing or canceling an appointment for a previously seen patient.		
Trigger: Patient calls and asks for a new appointment or asks to cancel or change an existing appointment		
Type: External		
Relationships: Association: Old Patient Include: Extend: Update Patient Information Generalization: Manage Appointments		
Normal Flow of Events: <ol style="list-style-type: none">1. The Patient contacts the office regarding an appointment.2. The Patient provides the Receptionist with his or her name and address.3. If the Patient's information has changed Execute the Update Patient Information use case.4. If the Patient's payment arrangements has changed Execute the Make Payments Arrangements use case.5. The Receptionist asks Patient if he or she would like to make a new appointment, cancel an existing appointment, or change an existing appointment. If the patient wants to make a new appointment, the S-1: new appointment subflow is performed. If the patient wants to cancel an existing appointment, the S-2: cancel appointment subflow is performed. If the patient wants to change an existing appointment, the S-3: change appointment subflow is performed.6. The Receptionist provides the results of the transaction to the Patient.		
SubFlows: S-1: New Appointment <ol style="list-style-type: none">1. The Receptionist asks the Patient for possible appointment times.2. The Receptionist matches the Patient's desired appointment times with available dates and times and schedules the new appointment. S-2: Cancel Appointment <ol style="list-style-type: none">1. The Receptionist asks the Patient for the old appointment time.2. The Receptionist finds the current appointment in the appointment file and cancels it. S-3: Change Appointment <ol style="list-style-type: none">1. The Receptionist performs the S-2: cancel appointment subflow.2. The Receptionist performs the S-1: new appointment subflow.		
Alternate/Exceptional Flows: S-1, 2a1: The Receptionist proposes some alternative appointment times based on what is available in the appointment schedule. S-1, 2a2: The Patient chooses one of the proposed times or decides not to make an appointment.		

Overview Information

- Identifies the use case and provides basic background information.
- The *use-case name* should be a verb–noun.
- The *use-case ID number* provides a unique way to find every use case and also enables the team to trace design decisions back to a specific requirement.
- The *use-case type* is either overview or detail and essential or real.
- The *primary actor* is usually the trigger of the use case—the person or thing that starts the execution of the use case. The primary purpose of the use case is to meet the goal of the primary actor.
- The *brief description* is typically a single sentence that describes the essence of the use case.

بازیگر اصلی معمولاً محرك است - شخص یا چیزی که اجرای مورد استفاده را آغاز می کند. هدف اصلی استفاده از این مورد، دستیابی به هدف بازیگر اصلی است. توصیف مختصر معمولاً یک جمله واحد است که ماهیت مورد استفاده را توصیف می کند.

Use Case Name: Make Old Patient Appt	ID: 2	Importance Level: Low
Primary Actor: Old Patient	Use Case Type: Detail, Essential	
Stakeholders and Interests:		
Old Patient – wants to make, change, or cancel an appointment Doctor – wants to ensure patient's needs are met in a timely manner		
Brief Description: This use case describes how we make an appointment as well as changing or canceling an appointment for a previously seen patient.		
Trigger: Patient calls and asks for a new appointment or asks to cancel or change an existing appointment		
Type: External		

Overview Information(Cnt'd)

- The *importance level* can be used to prioritize the use cases. The importance level enables the users to explicitly prioritize which business functions are most important and need to be part of the first version of the system and which are less important and can wait until later versions if necessary. The importance level can use a fuzzy scale, such as high, medium, and low. It can also be done more formally using a weighted average of a set of criteria.

از سطح اهمیت می توان برای اولویت بندی موارد استفاده کرد. سطح اهمیت به کاربران این امکان را می دهد که به صراحت اولویت بندی کنند که کدام عملکردهای تجاری مهم ترین هستند و باید بخشی از اولین نسخه سیستم باشند و کدام یک از اهمیت کمتری برخوردار هستند و در صورت لزوم می توانند تا نسخه های بعدی صبر کنند. سطح اهمیت می تواند از مقیاس فازی مانند بالا، متوسط و پایین استفاده کند. همچنین میتوان آن را با استفاده از میانگین وزنی مجموعهای از معیارها بهطور رسمیتر انجام داد

Use Case Name: Make Old Patient Appt	ID: 2	Importance Level: Low
Primary Actor: Old Patient	Use Case Type: Detail, Essential	
Stakeholders and Interests:		
Old Patient – wants to make, change, or cancel an appointment Doctor – wants to ensure patient's needs are met in a timely manner		
Brief Description: This use case describes how we make an appointment as well as changing or canceling an appointment for a previously seen patient.		
Trigger: Patient calls and asks for a new appointment or asks to cancel or change an existing appointment		
Type: External		

Overview Information(Cnt'd)

- A use case may have **multiple stakeholders** that have an interest in the use case. Each use case lists each of the stakeholders with each one's interest in the use case. The stakeholders' list always includes the primary actor.
- Each use case typically has a **trigger**—the event that causes the use case to begin. A trigger can be an **external trigger**, such as a customer placing an order or the fire alarm ringing, or it can be a **internal trigger**.

یک مورد استفاده ممکن است دارای ذینفعان متعددی باشد که در مورد استفاده علاقه دارند. هر مورد استفاده، هر یک از ذینفعان را با علاقه هر یک به مورد استفاده فهرست می کند. فهرست ذینفعان همیشه شامل بازیگر اصلی است.

هر مورد استفاده معمولاً یک ماشه دارد—رویدادی که باعث می شود مورد استفاده شروع شود. یک ماشه می تواند یک ماشه خارجی باشد، مانند سفارش مشتری یا زنگ هشدار آتش، یا می تواند یک ماشه داخلی باشد.

Use Case Name: Make Old Patient Appt	ID: 2	Importance Level: Low
Primary Actor: Old Patient	Use Case Type: Detail, Essential	
<p>Stakeholders and Interests:</p> <p>Old Patient – wants to make, change, or cancel an appointment Doctor – wants to ensure patient's needs are met in a timely manner</p>		
<p>Brief Description: This use case describes how we make an appointment as well as changing or canceling an appointment for a previously seen patient.</p>		
<p>Trigger: Patient calls and asks for a new appointment or asks to cancel or change an existing appointment</p>		
<p>Type: External</p>		

روابط مورد استفاده توضیح می دهد که چگونه مورد استفاده با سایر موارد استفاده و کاربران مرتبط است.

چهار نوع اساسی از روابط وجود دارد: ارتباط، گسترش، شامل و تعمیم •

یک رابطه ارتباطی، ارتباطی را که بین مورد استفاده و بازیگرانی که از مورد استفاده میکنند، صورت میگیرد، مستند میکند •

یک رابطه شامل گنجاندن اجباری مورد استفاده دیگری است. رابطه شامل تجزیه عملکردی را امکان پذیر می کند - تجزیه یک مورد استفاده پیچیده به چندین •

مورد ساده تر. رابطه شامل همچنین امکان استفاده مجدد از قسمتهایی از موارد استفاده را با ایجاد آنها به عنوان موارد استفاده جداگانه میدهد

Relationships

- Use-case relationships explain how the use case is related to other use cases and users.
- There are four basic types of relationships: **association**, **extend**, **include**, and **generalization**.
- An **association relationship** documents the communication that takes place between the use case and the actors that use the use case.
- An **include relationship** represents the mandatory inclusion of another use case. The include relationship enables **functional decomposition**—the breaking up of a complex use case into several simpler ones. The include relationship also enables parts of use cases to be reused by creating them as separate use cases.

Relationships(Cnt'd)

- An *extend relationship* represents the extension of the functionality of the use case to incorporate optional behavior.
- The *generalization relationship* allows use cases to support *inheritance*.

یک رابطه گسترش نشان دهنده گسترش عملکرد مورد استفاده برای ترکیب رفتار اختیاری است
رابطه تعمیم به موارد استفاده برای پشتیبانی از وراثت اجازه می دهد.

Flow of Events

- Finally, the individual steps within the business process are described. Three different categories of steps, or *flows of events*, can be documented: **normal flow of events**, **sub flows**, and **exceptional flows**.

در نهایت، مراحل فردی در فرآیند کسب و کار شرح داده شده است. سه دسته مختلف از مراحل یا جریان رویدادها را می توان مستند کرد: جریان عادی رویدادها، جریان های فرعی و جریان های استثنایی.

Normal flow of events

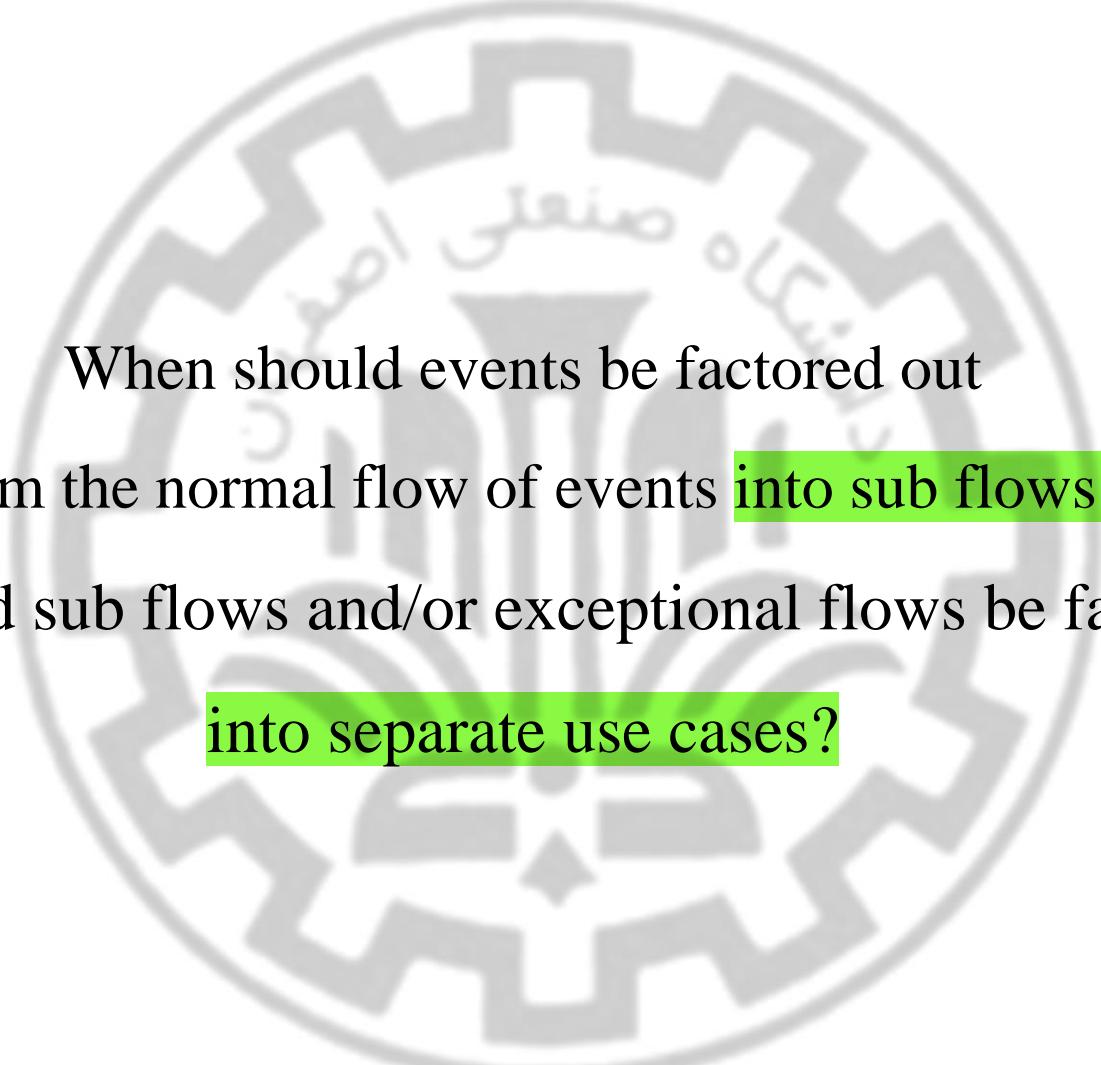
- Includes only steps that normally are executed in a use case. The steps are listed in the order in which they are performed.
- In some cases, the normal flow of events should be decomposed into a set of *sub flows* to keep the normal flow of events as simple as possible.
- Alternatively, we could replace a sub flow with a separate use case that could be incorporated via the include relationships. If it does, then the specific sub flow(s) should be replaced with a call to the related use case, and the use case should be added to the include relationship list.

فقط شامل مراحلی است که معمولاً در یک مورد استفاده اجرا می شوند. مراحل به ترتیبی که انجام می شوند فهرست شده اند. در برخی موارد، جریان عادی رویدادها باید به مجموعه ای از جریان های فرعی تجزیه شود تا جریان عادی رویدادها تا حد امکان ساده باشد. از طرف دیگر، میتوانیم یک جریان فرعی را با یک مورد استفاده جداگانه جایگزین کنیم که میتواند از طریق روابط شامل گنجانده شود. اگر اینطور باشد، جریان(های فرعی خاص باید با فرآخوانی به مورد استفاده مرتبط جایگزین شود و مورد استفاده باید به لیست ارتباط شامل اضافه شود)

Alternative or exceptional flows

- Are ones that do happen but are not considered to be the norm. These must be documented.
- Like the sub flows, the primary purpose of separating out alternate or exceptional flows is to keep the normal flow of events as simple as possible.

اینها مواردی هستند که اتفاق میافتد، اما عادی در نظر گرفته نمیشوند. اینها باید مستند باشد.
مانند جریان های فرعی، هدف اصلی از جداسازی جریان های متناوب یا استثنایی این است که
جریان عادی رویدادها تا حد امکان ساده باشد.



When should events be factored out
from the normal flow of events **into sub flows?**

When should sub flows and/or exceptional flows be factored out
into separate use cases?

The answer

- The primary criteria should be based on the level of complexity that the use case entails. The more difficult it is to understand the use case, the more likely events should be factored out into sub flows, or sub flows and/or alternative or exceptional flows should be factored out into separate use cases that are called by the current use case. This, creates more use cases.
- We are trying to represent, in a manner as complete and concise as possible, our understanding of the business processes that we are investigating so that the client can validate the requirements that we are modeling.

معیارهای اولیه باید بر اساس سطح پیچیدگی مورد استفاده باشد. هرچه درک مورد استفاده دشوارتر باشد، رویدادهای محتملتری باید در جریانهای فرعی در نظر گرفته شوند، یا جریانهای فرعی و/یا جریانهای جایگزین یا استثنایی باید در موارد استفاده جداگانه که توسط مورد استفاده فعلی فراخوانی میشوند، لحاظ شوند. این، موارد استفاده بیشتری را ایجاد می کند. ما تلاش می کنیم تا حد امکان کامل و مختصر درک خود را از فرآیندهای تجاری که در حال بررسی آن ها هستیم نشان دهیم تا مشتری بتواند الزاماتی را که ما در حال مدل سازی هستیم تأیید کند.

Optional Characteristics

- Other characteristics of use cases can be documented by use-case descriptions.
 - Level of complexity of the use case;
 - The estimated amount of time it takes to execute the use case;
 - The system with which the use case is associated;
 - Specific data flows between the primary actor and the use case;
 - Any specific attribute, constraint, or operation associated with the use case;
 - Any preconditions that must be satisfied for the use case to execute;
 - Any guarantees that can be made based on the execution of the use case.
- There is no standard set of characteristics of a use case that must be captured.

سایر ویژگی های موارد استفاده را می توان با
شرح موارد استفاده مستند کرد
سطح پیچیدگی مورد استفاده •
مدت زمان تخمینی که برای اجرای case
case طول می کشد •
سیستمی که مورد استفاده با آن مرتبط است •
جریان داده های خاص بین بازیگر اصلی و
موردن استفاده •
هر ویژگی خاص، محدودیت، یا عملیات مرتبط •
با مورد استفاده •
هر پیش شرطی که باید برای اجرا مورد استفاده
قرار گیرد •
هر گونه تضمینی که بتوان بر اساس اجرای
موردن استفاده انجام داد •

Campus Housing Service Add an Apartment

Use-Case Description

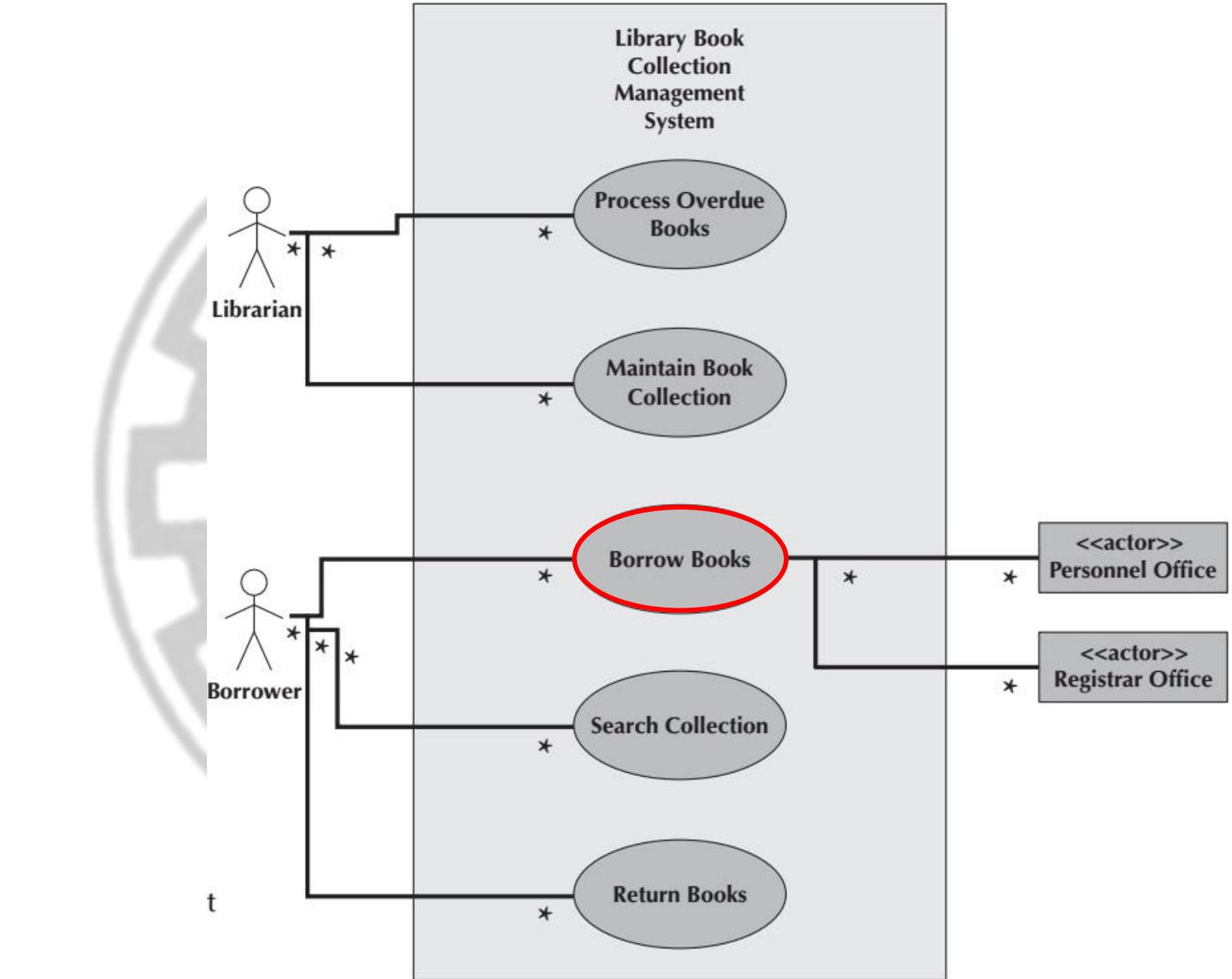
Use Case Name: Add Apartment	ID: 1	Importance Level: High
Primary Actor: Apartment Owner	Use Case Type: Detail, Essential	
<p>Stakeholders and Interests:</p> <p>Apartment Owner—wants to advertise available apartment Campus Housing Service—provides a service that enables the apartment owners to rent their available apartments</p>		
<p>Brief Description: This use case describes how the campus housing service can maintain an up-to-date listing of available apartments.</p>		
<p>Trigger: Apartment Owner wants to add an available apartment</p>		
<p>Type: External</p>		
<p>Relationships:</p> <p>Association: Apartment Owner Include: Extend: Generalization:</p>		
<p>Normal Flow of Events:</p> <ol style="list-style-type: none">1. Capture the location of the apartment.2. Capture the number of bedrooms in the apartment.3. Capture the monthly rent of the apartment.4. Add the apartment to the listing of available apartments.		
<p>SubFlows:</p>		
<p>Alternate/Exceptional Flows:</p>		

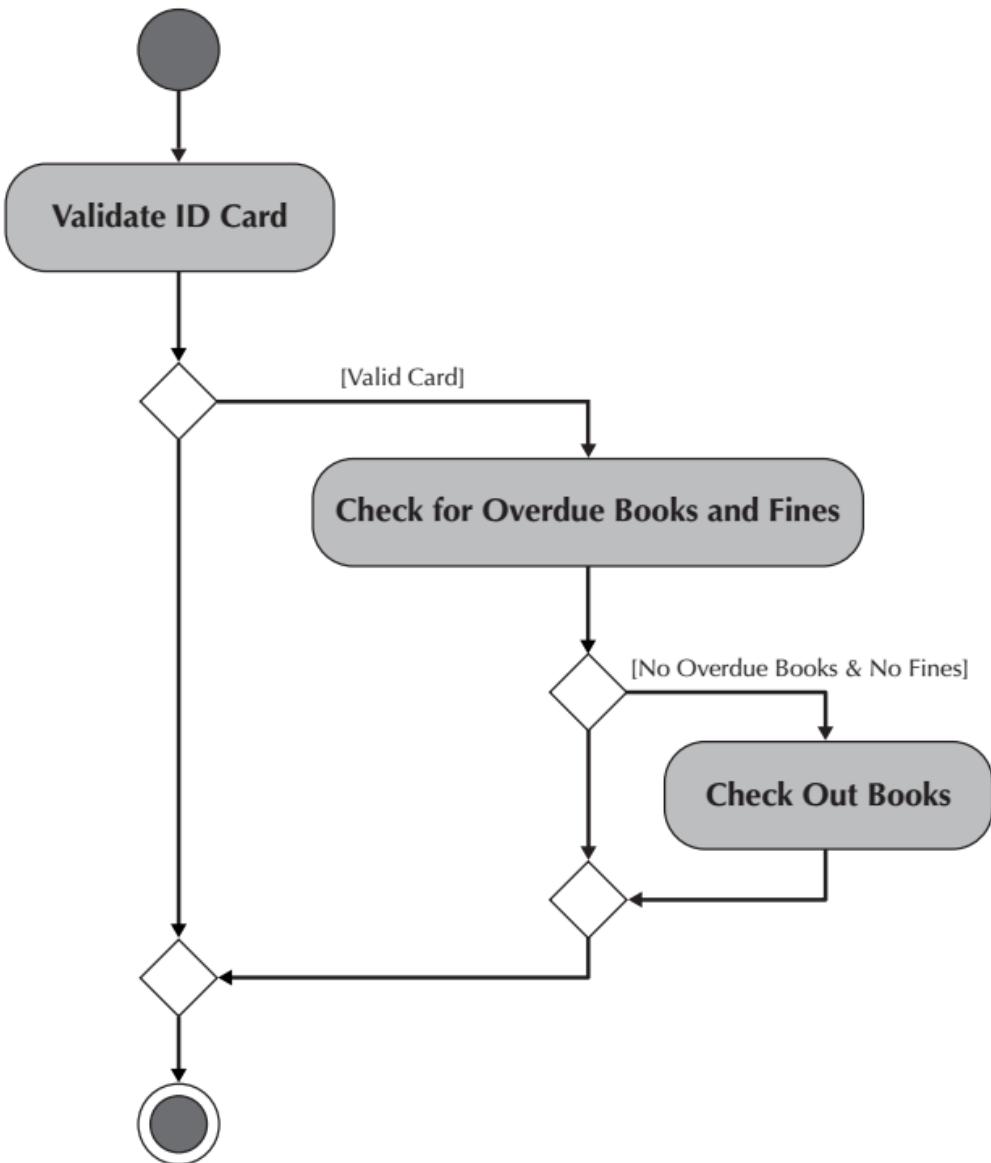
Campus Housing Service

Delete an Apartment

Use-Case Description

Use Case Name:	Delete Apartment	ID:	2	Importance Level:	High
Primary Actor:	Apartment Owner	Use Case Type:	Detail, Essential		
Stakeholders and Interests:					
Apartment Owner—wants to delist apartment Campus Housing Service—provides a service that enables the apartment owners to rent their available apartments					
Brief Description: This use case describes how the campus housing service can maintain an up-to-date listing of available apartments.					
Trigger: Apartment Owner wants to delete an available apartment					
Type: External					
Relationships:					
Association: Apartment Owner Include: Extend: Generalization:					
Normal Flow of Events:					
1. Capture the apartment identifier. 2. Delete the apartment from the listing of available apartments.					
SubFlows:					
Alternate/Exceptional Flows:					





Overview Description for the Borrow Books Use Case

Use Case Name: Borrow Books	ID: 2	Importance Level: <u>High</u>		
Primary Actor: Borrower	Use Case Type: Detail, Essential			
<p>Stakeholders and Interests:</p> <p>Borrower—wants to check out books</p> <p>Librarian—wants to ensure borrower only gets books deserved</p>				
<p>Brief Description: This use case describes how books are checked out of the library.</p>				
<p>Trigger: Borrower brings books to check out desk.</p>				
<p>Type: External</p>				
<p>Relationships:</p> <p>Association: Borrower, Personnel Office, Registrar's Office</p> <p>Include:</p>				
<p>Extend:</p> <p>Generalization:</p>				

Flow Descriptions for the Borrow Books Use Case

Normal Flow of Events:

1. The Borrower brings books to the Librarian at the check out desk.
2. The Borrower provides Librarian their ID card.
3. The Librarian checks the validity of the ID Card.
 - If the Borrower is a Student Borrower, Validate ID Card against Registrar's Database.
 - If the Borrower is a Faculty/Staff Borrower, Validate ID Card against Personnel Database.
 - If the Borrower is a Guest Borrower, Validate ID Card against Library's Guest Database.
4. The Librarian checks whether the Borrower has any overdue books and/or fines
5. The Borrower checks out the books

SubFlows:

Alternate/Exceptional Flows:

- 4a The ID Card is invalid, the book request is rejected.
- 5a The Borrower either has overdue books, fines, or both, the book request is rejected.

What should you do for your project?

1. Write use case description for each use case.

We will work in the lab.

Reference

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

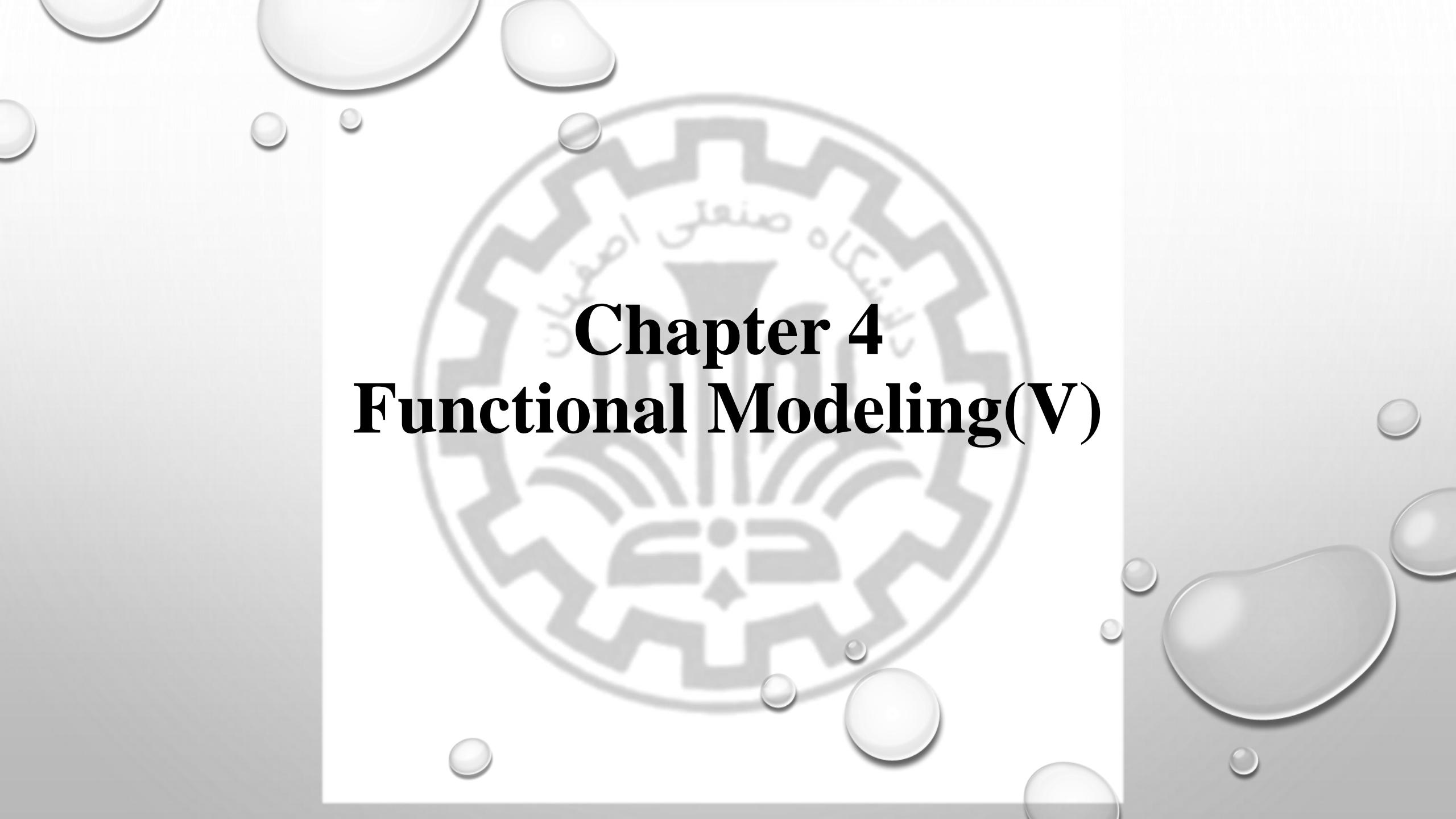
Software Engineering I

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2021



Chapter 4

Functional Modeling(V)

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

Verifying and Validating Functional Models

- **Walkthroughs:** is essentially a peer review of a product. In the case of the functional models, a walkthrough is a review of the different models and diagrams created during functional modeling.
- The purpose of a walkthrough is to thoroughly *test* the fidelity of the functional models to the functional requirements and to ensure that the models are **consistent**. That is, a walkthrough uncovers *errors* or *faults* in the evolving specification.

اساساً یک بررسی همکار از یک محصول است. در: **Walkthroughs:** مورد مدل‌های عملکردی، مروری بر مدل‌ها و نمودارهای مختلف ایجاد شده در طول مدلسازی عملکردی است.

هدف از یک راهپیمایی آزمایش کامل وفاداری مدل‌های عملکردی به الزامات عملکردی و اطمینان از سازگاری مدل‌ها است. به این معنا که یک بررسی، خطاهای عیوب در مشخصات در حال تکامل را آشکار می‌کند

Walkthrough

- Walkthroughs are very interactive. As the presenter walks through the representation, members of the walkthrough team should ask questions regarding the representation.
- For example, if the presenter is walking through an activity diagram, another member of the team could ask why certain activities or objects were not included.
- The actual process of simply presenting the representation to a new set of eyes can uncover obvious misunderstandings and omissions.

راهنماها بسیار تعاملی هستند. همانطور که ارائه دهنده از طریق نمایندگی عبور می کند، اعضای تیم راهنما باید سوالاتی در مورد نمایندگی بپرسند.

به عنوان مثال، اگر ارائه دهنده در حال عبور از نمودار فعالیت است، یکی دیگر از اعضای تیم می تواند بپرسد که چرا فعالیت ها یا اشیاء خاصی گنجانده نشده اند

فرآیند واقعی ارائه صرفاً نمایش به مجموعه ای از چشمان جدید می تواند سوء تفاهem ها و حذفیات آشکار را آشکار کند.

Roles

- There are specified roles that different members of the walkthrough team can play.
- *Presenter*: should be played by the person who is primarily responsible for the specific representation being reviewed.
- *recorder, or scribe*: should be a member of the analysis team. This individual carefully takes the minutes of the meeting by recording all significant events that occur during the walkthrough. In particular, all errors that are uncovered must be documented so that the analysis team can address them.

ارائه کننده: باید توسط شخصی که مسئول اصلی نمایش خاص در حال بررسی است بازی شود •
ضبط کننده یا کاتب: باید عضوی از تیم تحلیل باشد. این فرد با ثبت تمام وقایع مهمی که در طول بازدید رخ می دهد، با دقت صورتجلسات جلسه را می گیرد. به ویژه، تمام خطاهایی که کشف می شوند باید مستند شوند تا تیم تجزیه و تحلیل بتواند به آنها رسیدگی کند

Functional Model Verification and Validation

- First, when comparing an activity diagram to a use-case description, there should be at least one event recorded in the normal flow of events, subflows, or alternative/exceptional flows of the use-case description for each activity or action that is included on an activity diagram, and each event should be associated with an activity or action.
- Second, all objects portrayed as an object node in an activity diagram must be mentioned in an event in the normal flow of events, subflows, or alternative/exceptional flows of the use-case description.
- Third, sequential order of the events in a use-case description should occur in the same sequential order of the activities contained in an activity diagram.

Functional Model Verification and Validation(Cnt'd)

- Fourth, when comparing a use-case description to a use-case diagram, there must be one and only one use-case description for each use case, and vice versa.
- Fifth, all actors listed in a use-case description must be portrayed on the use-case diagram. Each actor must have an association link that connects it to the use case and must be listed with the association relationships in the use-case description.
- Sixth, in some organizations, we should also include the stakeholders listed in the use-case description as actors in the use-case diagram.
- Seventh, all other relationships listed in a use-case description (include, extend, and generalization) must be portrayed on a use-case diagram.

Reference

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

Software Engineering I

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2021

Chapter 5

Structural Modeling

Steps(I)

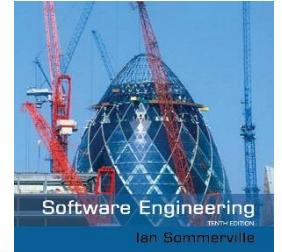
1. Preparing proposal
2. Requirements determination
 - User story
3. Abstract Business Process Modelling
4. Analysis
 - Functional Modelling
 - Structural Modelling
 - Behavioral Modelling

Steps(II)

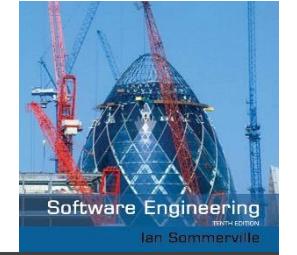
5. Design

- Optimization
- Database Management
- User Interface
- Physical Architecture

System modeling

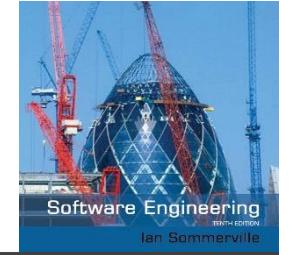


- ✧ System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- ✧ System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- ✧ System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.



System perspectives

- ✧ An external perspective, where you model the context or environment of the system.
- ✧ A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- ✧ A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.



UML diagram types

- ✧ Use case diagrams, which show the interactions between a system and its environment.
- ✧ Activity diagrams, which show the activities involved in a process or in data processing .
- ✧ Class diagrams, which show the object classes in the system and the associations between these classes.
- ✧ Sequence diagrams, which show interactions between actors and the system and between system components.
- ✧ State diagrams, which show how the system reacts to internal and external events.

Structural modeling

- Supports the creation of an **internal structural** or **static view** of a business information system in that it shows how the system is structured to support the underlying business processes.
- A **structural model** is a **formal way of representing the objects** that are used and created by a business system. It illustrates **people, places, or things** about which information is captured and **how they are related to one another**.
- The structural model is drawn using an **iterative process** in which the model becomes more detailed over time.

از ایجاد یک نمای ساختاری یا ایستادخانی از یک سیستم اطلاعات کسب و کار پشتیبانی می‌کند، زیرا نشان می‌دهد که چگونه سیستم برای پشتیبانی از فرآیندهای تجاری زیربنایی ساختار یافته است. مدل ساختاری روشی رسمی برای نمایش اشیایی است که توسط یک سیستم تجاری استفاده و ایجاد می‌شود. این افراد، مکانها یا چیزهایی را نشان میدهد که اطلاعات مربوط به آنها و نحوه ارتباط آنها با یکدیگر چیست. مدل ساختاری با استفاده از یک فرآیند تکراری ترسیم می‌شود که در آن مدل با گذشت زمان جزئیات بیشتری پیدا می‌کند.

در تجزیه و تحلیل، تحلیلگران یک مدل مفهومی ترسیم می کنند، که سازماندهی منطقی اشیاء را بدون نشان دادن نحوه ذخیره، ایجاد یا دستکاری اشیاء نشان می دهد. از آنجایی که این مدل عاری از هرگونه پیاده سازی یا جزئیات فنی است، تحلیلگران می توانند به راحتی روی تطبیق مدل با الزامات تجاری واقعی سیستم تمرکز کنند.

Structural modelling

- In analysis, analysts draw a **conceptual model**, which shows the **logical** organization of the objects **without** indicating **how** the objects are stored, created, or manipulated. Because this model is **free from any implementation or technical details**, the analysts can focus more easily on matching the model to the **real business requirements** of the system.
- In design, analysts evolve the conceptual structural model into **a design model** that reflects how the **objects** will be organized **in databases and software**. At this point, the model is checked for **redundancy**, and the analysts investigate ways to make the objects **easy to retrieve**.

در طراحی، تحلیلگران مدل ساختاری مفهومی را به یک مدل طراحی تبدیل میکنند که نحوه سازماندهی اشیاء در پایگاههای داده و نرمافزار را نشان میدهد. در این مرحله، مدل از نظر افزونگی بررسی میشود و تحلیلگران راههایی را بررسی میکنند تا اشیا را به راحتی بازیابی کنند.

Structural Models

- The goal of the analyst is to discover the **key objects** contained in the **problem domain** and to build a structural model.
- Basic elements of structural models are **classes**, **attributes**, **operations**, and **relationships**.

هدف تحلیلگر این است که اشیاء کلیدی موجود در حوزه مسئله را کشف کند و یک مدل ساختاری بسازد.
عناصر اساسی مدل های ساختاری کلاس ها، ویژگی ها، عملیات و روابط هستند.

Class

کلاس یک الگوی کلی است که ما از آن برای ایجاد نمونه ها یا اشیاء خاص در حوزه مشکل استفاده می کنیم.

- تمام اشیاء یک کلاس معین از نظر ساختار و رفتار یکسان هستند اما دارای داده های متفاوتی در ویژگی های خود هستند
- دو نوع کلی از کلاس های مورد علاقه در هنگام تحلیل وجود دارد: عینی و انتزاعی
- کلاس های بتن برای ایجاد اشیا استفاده می شود
- کلاس های انتزاعی در واقع در دنیای واقعی وجود ندارند. آنها صرفاً انتزاعات مفیدی هستند

- A *class* is a **general template** that we use to create specific **instances**, or **objects**, in the problem domain.
- All objects of a given class are **identical in structure and behavior** but contain **different data** in their **attributes**.
- There are **two general kinds of classes** of interest during analysis: **concrete** and **abstract**.
 - *Concrete classes* are used to **create objects**.
 - *Abstract classes* do not actually exist in the real world; they are simply useful abstractions.

A second classification of classes

- Is the type of real-world thing that a class represents.
 - domain classes,
 - user-interface classes,
 - data structure classes,
 - file structure classes,
 - operating environment classes,
 - document classes,
 -

Class

- An **attribute** of an analysis class represents a **piece of information** that is relevant to the description of the class **within the application domain** of the problem being investigated.
- The **behavior** of an analysis class is defined in an **operation or service**. In later phases, the operations are converted to **methods**.

یک ویژگی یک کلاس تجزیه و تحلیل، بخشی از اطلاعات را نشان می دهد که مربوط به توصیف کلاس در حوزه کاربردی مسئله مورد بررسی است.
رفتار یک کلاس تجزیه و تحلیل در یک عملیات یا سرویس تعریف می شود. در
مراحل بعدی، عملیات به روش تبدیل می شود.

Object Identification

- **Textual Analysis**
- **Brainstorming**
- **Common Object Lists**
- **Patterns**

- تحلیل متن
- طوفان فکری
- فهرست اشیاء مشترک
- الگوهای

CRC Cards

- *CRC (Class–Responsibility–Collaboration) cards* are used to document the responsibilities and collaborations of a class.
- *Responsibilities* of a class can be broken into two separate types: **knowing** and **doing**.
 - *Knowing responsibilities* are those things that an instance of a class must be capable of knowing. An instance of a class typically **knows the values of its attributes and its relationships**.
 - *Doing responsibilities* are those things that an instance of a class must be capable of doing. In this case, an instance of a class **can execute its operations**.

برای مستندسازی مسئولیت‌ها و همکاری‌های یک کلاس استفاده می‌شوند **CRC** کارت‌های مسئولیت‌های یک کلاس را می‌توان به دو نوع مجزا تقسیم کرد: دانستن و انجام دادن •
دانستن مسئولیت‌ها چیزهایی هستند که یک نمونه از یک کلاس باید قادر به دانستن آنها باشد. نمونه‌ای از یک کلاس معمولاً •
مقادیر ویژگی‌ها و روابط آن را می‌داند
• انجام مسئولیت‌ها آن دسته از کارهایی هستند که یک نمونه از یک کلاس باید قادر به انجام آنها باشد. در این حالت، یک نمونه •
از یک کلاس می‌تواند عملیات خود را اجرا کند

CRC Cards(Cnt'd)

- **Collaborations** allow the analyst to think in terms of clients, servers, and contracts.
 - A *client object* is an instance of a class that **sends a request** to an instance of another class for an operation to be executed.
 - A *server object* is the instance that **receives the request** from the client object.
 - A *contract* formalizes the interactions between the client and server objects.

همکاریها به تحلیلگر اجازه میدهد تا در مورد مشتریان، سرورها و قراردادها فکر کند.
یک شی کلاینت نمونه‌ای از یک کلاس است که درخواستی را به نمونه‌ای از کلاس دیگر ارسال می‌کند تا عملیاتی اجرا شود.
یک شی سرور نمونه‌ای است که درخواست را از شی مشتری دریافت می‌کند.
یک قرارداد تعاملات بین اشیاء مشتری و سرور را رسمی می‌کند.

Elements of a CRC Card

- The front of the card contains the class's name, ID, type, description, associated use cases, responsibilities, and collaborators.
- The back of a CRC card contains the attributes and relationships of the class. The attributes of the class represent the knowing responsibilities that each instance of the class has to meet.

جلوی کارت شامل نام کلاس، شناسه، نوع، توضیحات، موارد استفاده مرتبط، مسئولیتها و همکاران است.
حاوی ویژگی ها و روابط کلاس است. ویژگی های کلاس نشان دهنده مسئولیت های CRC پشت کارت
دانشی است که هر نمونه از کلاس باید انجام دهد.

Front:

Class Name: Old Patient	ID: 3	Type: Concrete, Domain
Description: An individual who needs to receive or has received medical attention		Associated Use Cases: 2
Responsibilities		Collaborators
Make appointment		Appointment
Calculate last visit		
Change status		
Provide medical history		Medical history

Back:

Attributes:

Amount (double)

Insurance carrier (text)

Relationships:

Generalization (a-kind-of): Person

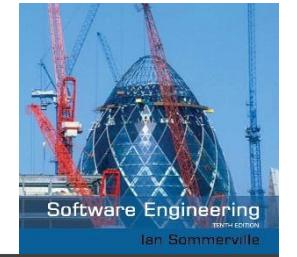
Aggregation (has-parts): Medical History

Other Associations: Appointment

Class Diagrams

- A *class diagram* is a *static model* that shows the classes and the relationships among classes that remain **constant** in the system over time.
- Elements of a Class Diagram
 - **Class:** The main building block of a class diagram is the class, which stores and manages information in the system. Visibility relates to the **level** of information hiding to be enforced for the attribute. Visibility of an attribute can be **public (+)**, **protected (#)**, or **private (-)**.

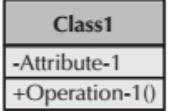
نمودار کلاس یک مدل استاتیک است که طبقات و روابط بین طبقاتی را نشان می دهد که در طول زمان در سیستم ثابت می مانند.



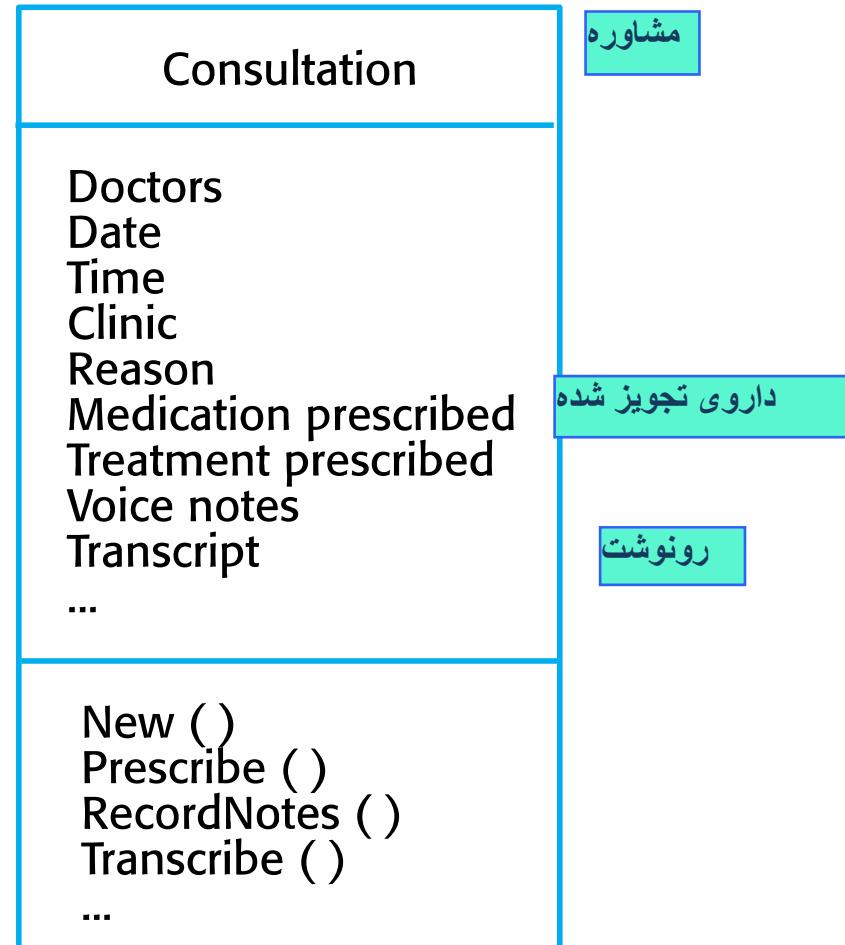
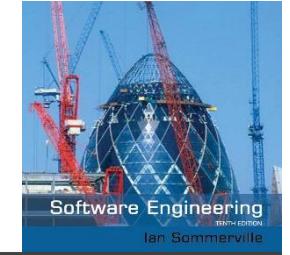
Class diagrams

- ✧ Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- ✧ An object class can be thought of as a general definition of one kind of system object.
- ✧ An association is a link between classes that indicates that there is some relationship between these classes.
- ✧ When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.

نمودارهای کلاس هنگام توسعه یک مدل سیستم شی گرا برای نشان دادن کلاس‌ها در یک سیستم و ارتباط بین این کلاس‌ها استفاده می‌شود. یک کلاس شی را می‌توان به عنوان یک تعریف کلی از یک نوع شی سیستم در نظر گرفت. ارتباط پیوندی بین کلاس‌ها است که نشان می‌دهد رابطه‌ای بین این کلاس‌ها وجود دارد. هنگامی که در مراحل اولیه فرآیند مهندسی نرم افزار در حال توسعه مدل هستید، اشیا چیزی را در دنیای واقعی نشان می‌دهند، مانند یک بیمار، یک نسخه، پزشک و غیره.

A class: <ul style="list-style-type: none">• Represents a kind of person, place, or thing about which the system will need to capture and store information.• Has a name typed in bold and centered in its top compartment.• Has a list of attributes in its middle compartment.• Has a list of operations in its bottom compartment.• Does not explicitly show operations that are available to all classes.	
An attribute: <ul style="list-style-type: none">• Represents properties that describe the state of an object.• Can be derived from other attributes, shown by placing a slash before the attribute's name.	attribute name /derived attribute name
An operation: <ul style="list-style-type: none">• Represents the actions or functions that a class can perform.• Can be classified as a constructor, query, or update operation.• Includes parentheses that may contain parameters or information needed to perform the operation.	operation name ()
An association: <ul style="list-style-type: none">• Represents a relationship between multiple classes or a class and itself.• Is labeled using a verb phrase or a role name, whichever better represents the relationship.• Can exist between one or more classes.• Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance.	
A generalization: <ul style="list-style-type: none">• Represents a-kind-of relationship between multiple classes.	
An aggregation: <ul style="list-style-type: none">• Represents a logical a-part-of relationship between multiple classes or a class and itself.• Is a special form of an association.	
A composition: <ul style="list-style-type: none">• Represents a physical a-part-of relationship between multiple classes or a class and itself• Is a special form of an association.	

The Consultation class



Relationships

- Generalization

- Enables the analyst to create classes that inherit attributes and operations of other classes. The subclasses inherit the attributes and operations of their superclass and can also contain attributes and operations that are unique just to them.
- Is represented with the *a-kind-of* relationship, so that we say that an employee is a-kind-of person.

- Aggregation

- Relate *parts to wholes*. For our purposes, we use the *a-part-of* or *has-parts* semantic relationship to represent the aggregation abstraction. For example, a door is a-part-of a car.

• تعمیم

تحلیلگر را قادر می سازد تا کلاس هایی ایجاد کند که ویژگی ها و عملیات کلاس های دیگر را به ارث می برند. زیر کلاسها ویژگیها و عملیات سوپرکلاس خود را به ارث میبرند و همچنین میتوانند شامل ویژگیها و عملیاتهایی باشند که فقط مختص آنهاست.
با رابطه یک نوع نشان داده می شود، به طوری که می گوییم یک کارمند یک نوع فرد است.

• تجمع

برای نمایش انتزاع تجمع استفاده می کنیم. به عنوان مثال، یک درب a-part-of has-parts را به کل ربط دهد. برای اهداف خود، از رابطه معنایی جزئی از یک ماشین است

Relationships(Cnt'd)

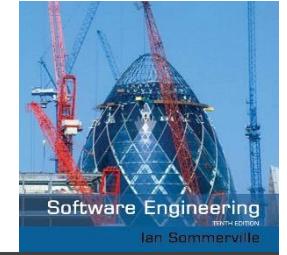
• Association

- There are other types of relationships that do not fit neatly into a generalization (a-kind-of) or aggregation (a-part-of) framework.
- Thus, they are simply considered to be *associations* between instances of classes.

• اتحادیه
• انواع دیگری از روابط وجود دارد که به طور منظم در چارچوب تعمیم (نوعی) یا تجمعی قرار نمی گیرند (a-part-of). بنابراین، آنها صرفاً به عنوان ارتباط بین نمونه هایی از کلاس ها در نظر گرفته می شوند.

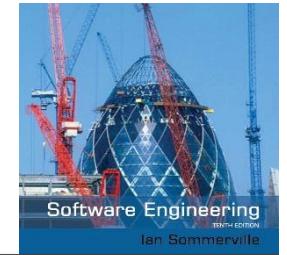


Generalization



- ✧ Generalization is an everyday technique that we use to manage complexity.
- ✧ Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes.
- ✧ This allows us to infer that different members of these classes have some common characteristics e.g. squirrels and rats are rodents.

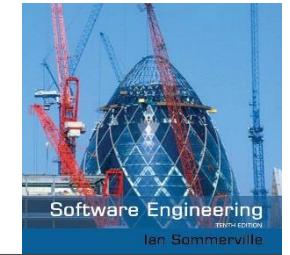
تعمیم یک تکنیک روزمره است که ما از آن برای مدیریت پیچیدگی استفاده می کنیم. به جای یادگیری ویژگی های جزئی هر موجودیتی که تجربه می کنیم، این موجودیت ها را در طبقات عمومی تر (حیوانات، ماشین ها، خانه ها و غیره) قرار می دهیم و ویژگی های این طبقات را می آموزیم. این به ما امکان می دهد استنباط کنیم که اعضای مختلف این کلاس ها دارای برخی ویژگی های مشترک هستند به عنوان مثال. سنجاب ها و موش ها جوندگان هستند.



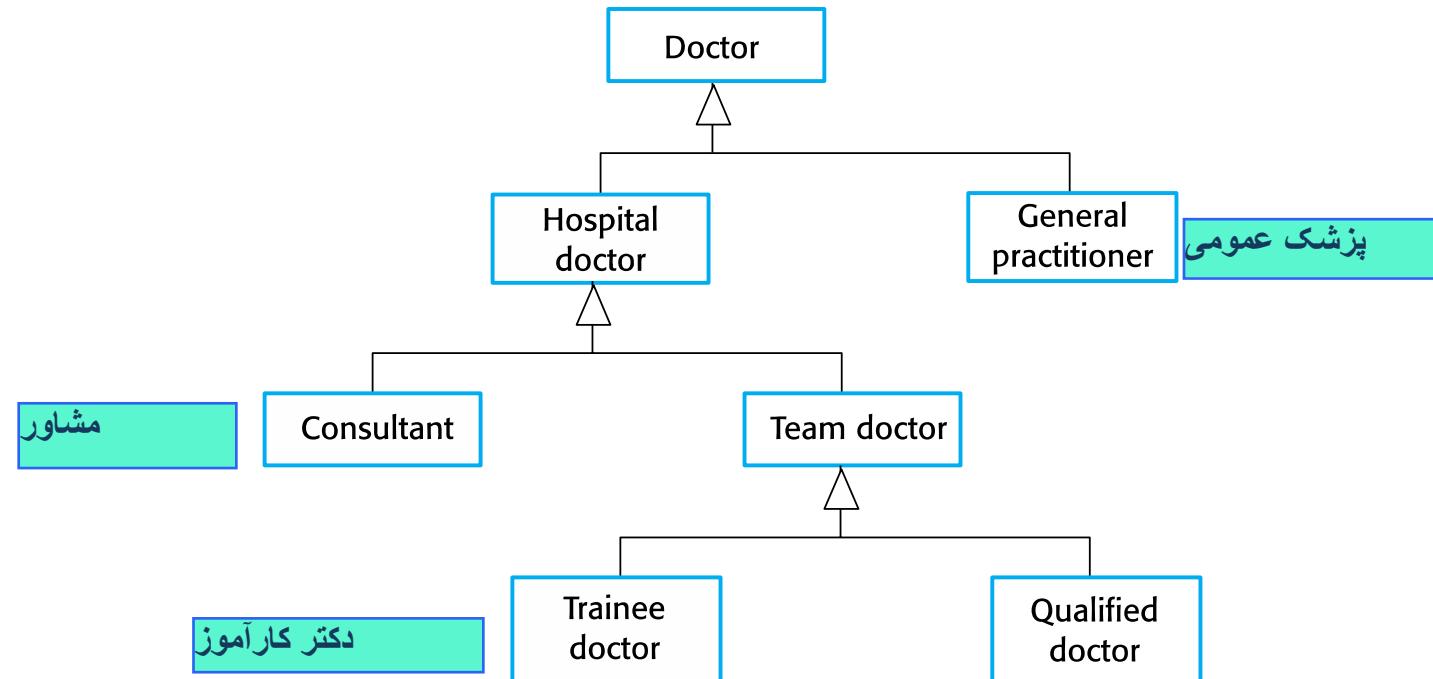
Generalization

- ✧ In modeling systems, it is often useful to examine the classes in a system to see if there is **scope for generalization**. If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change.
- ✧ In object-oriented languages, such as Java, generalization is implemented using the **class inheritance mechanisms** built into the language.
- ✧ In a generalization, the **attributes and operations** associated with higher-level classes are also associated with the lower-level classes.
- ✧ The lower-level classes are subclasses **inherit the attributes and operations** from their superclasses. These lower-level classes then **add more specific attributes and operations**.

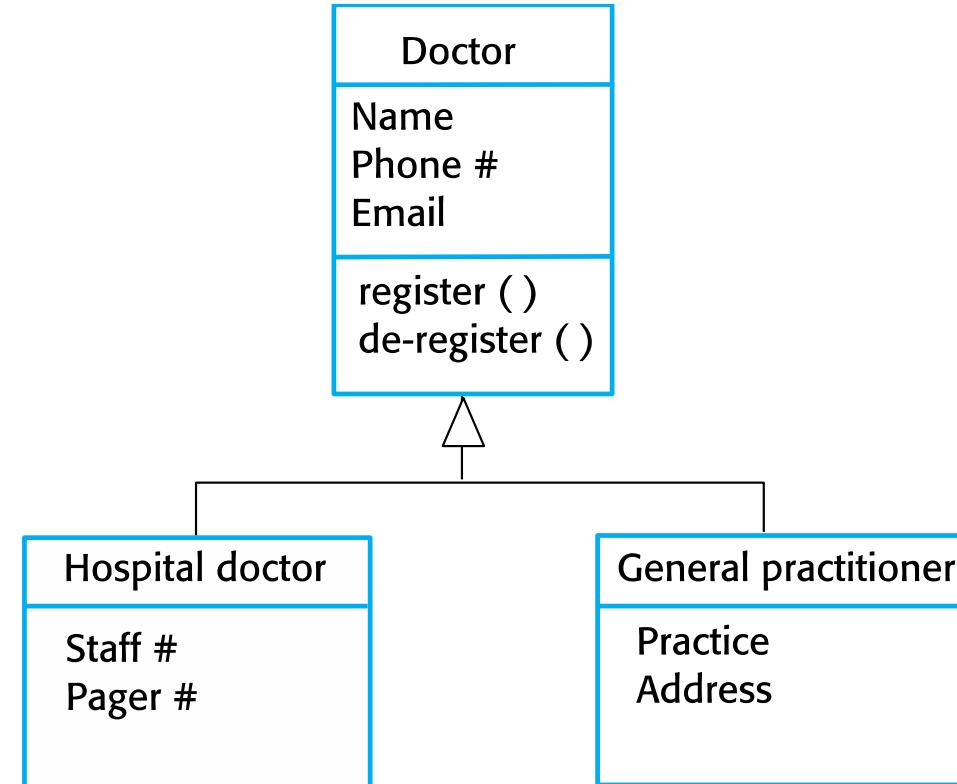
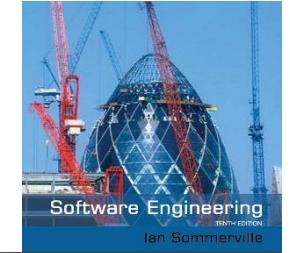
A generalization hierarchy

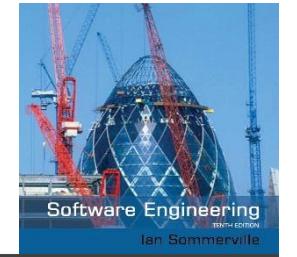


Software Engineering
Ian Sommerville



A generalization hierarchy with added detail



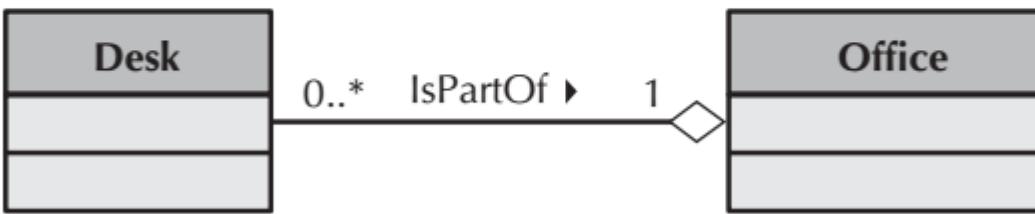
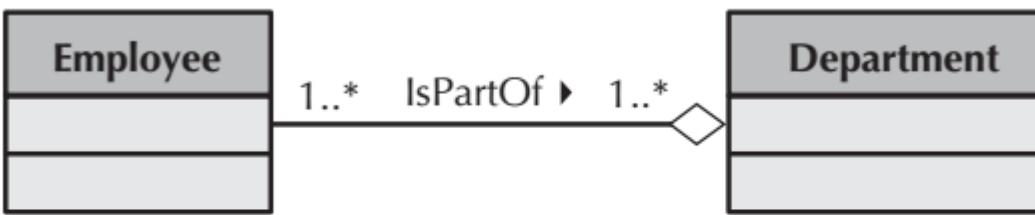


Object class aggregation models

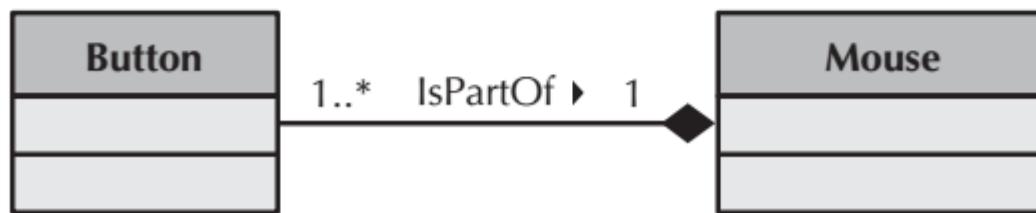
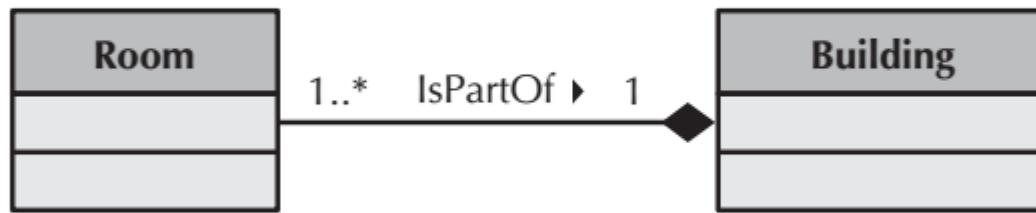
- ✧ An **aggregation model** shows how classes that are collections are composed of other classes.
- ✧ Aggregation models are similar to the **part-of relationship** in semantic data models.

یک مدل تجمعی نشان می دهد که چگونه کلاس هایی که مجموعه هستند از کلاس های دیگر تشکیل شده اند.
مدل های تجمعی شبیه به رابطه بخشی در مدل های داده های معنایی هستند.

Sample Aggregation Associations

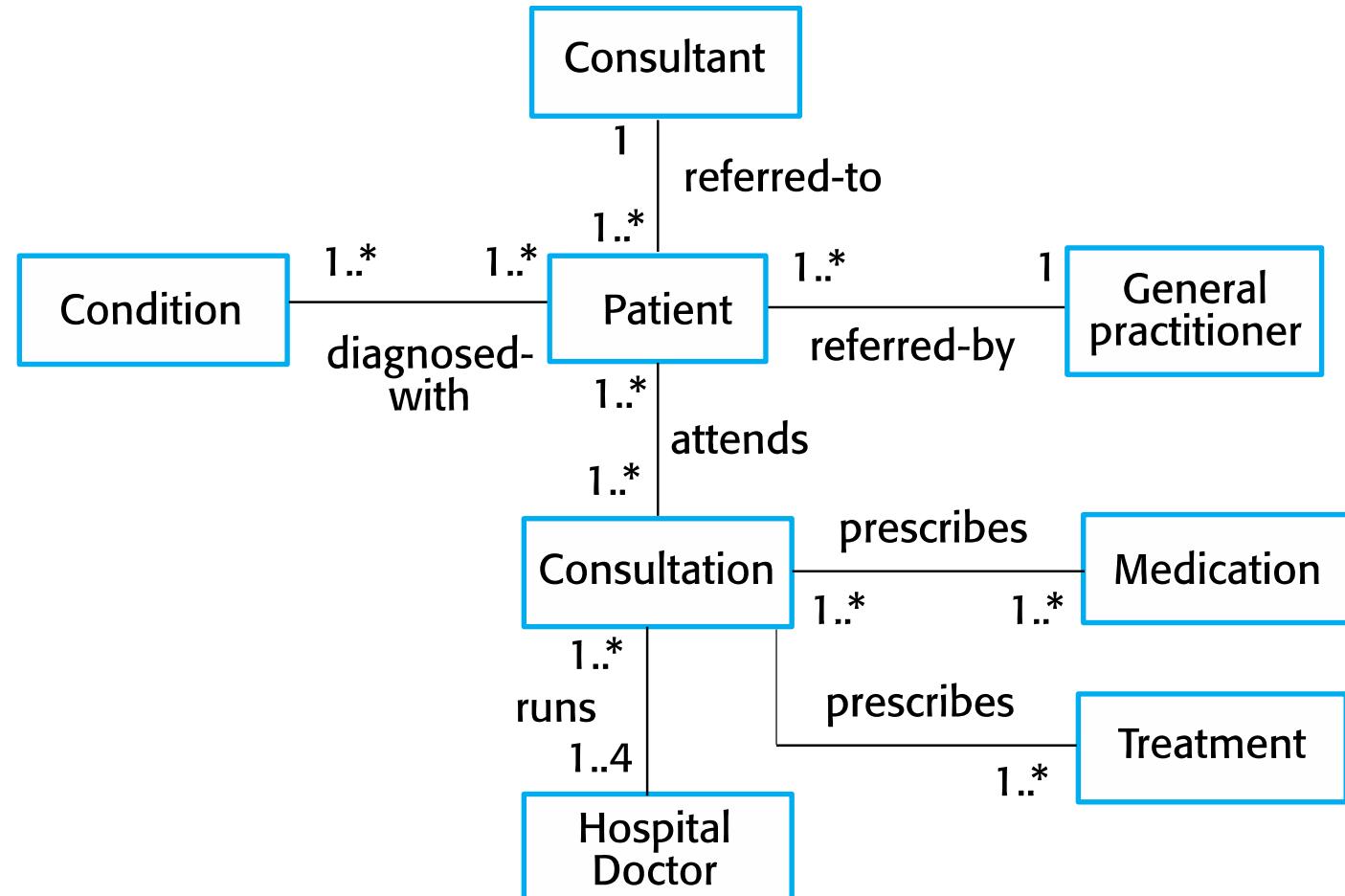
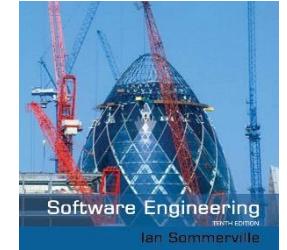


Sample Composition Associations



Exactly one	1	<pre> classDiagram class Department class Boss Department "1" --> "1" Boss </pre>	A department has one and only one boss.
Zero or more	0..*	<pre> classDiagram class Employee class Child Employee "0..*" --> "0..*" Child </pre>	An employee has zero to many children.
One or more	1..*	<pre> classDiagram class Boss class Employee Boss "1..*" --> "1..*" Employee </pre>	A boss is responsible for one or more employees.
Zero or one	0..1	<pre> classDiagram class Employee class Spouse Employee "0..1" --> "0..1" Spouse </pre>	An employee can be married to zero or one spouse.
Specified range	2..4	<pre> classDiagram class Employee class Vacation Employee "2..4" --> "2..4" Vacation </pre>	An employee can take from two to four vacations each year.
Multiple, disjoint ranges	1..3,5	<pre> classDiagram class Employee class Committee Employee "1..3,5" --> "1..3,5" Committee </pre>	An employee is a member of one to three or five committees.

Classes and associations in the MHC-PMS



Verifying And Validating The Structural Model

- Accomplished during a formal review meeting using a walkthrough approach in which an analyst presents the model to a team of developers and users. The analyst walks through the model, explaining each part of the model and all the reasoning behind the decision to include each of the classes in the structural model.

در طی یک جلسه بررسی رسمی با استفاده از یک رویکرد راهنمایی در آن یک تحلیلگر مدل را به تیمی از توسعه دهندگان و کاربران ارائه می دهد، انجام شد. تحلیلگر مدل را طی می کند و هر بخش از مدل و همه استدلال های پشت تصمیم را برای گنجاندن هر یک از کلاس ها در مدل ساختاری توضیح می دهد.

Verifying And Validating The Structural Model(Cnt'd)

- Test the consistency within the structural models.
 - First, every CRC card should be associated with a class on the class diagram, and vice versa.
 - Second, the responsibilities listed on the front of the CRC card must be included as operations in a class on a class diagram, and vice versa.
 - Third, collaborators on the front of the CRC card imply some type of relationship on the back of the CRC card and some type of association that is connected to the associated class on the class diagram.
 - Fourth, attributes listed on the back of the CRC card must be included as attributes in a class on a class diagram, and vice versa.
 - Fifth, the object type of the attributes listed on the back of the CRC card and with the attributes in the attribute list of the class on a class diagram implies an association from the class to the class of the object type.

Verifying And Validating The Structural Model(Cnt'd)

- Test the consistency within the structural models.
 - Sixth, the **relationships** included on the **back** of the CRC card must be portrayed using the **appropriate notation** on the class diagram.
 - Seventh, an **association class** should be created only if there is indeed **some unique characteristic** (attribute, operation, or relationship) about the intersection of the connecting classes.

What should you do for your project?

1. Create class diagram.



We will work in the lab.

Reference

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

Software Engineering I

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2021

Chapter 6

Behavioral modeling(I)

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

Behavioral model

- Describe the internal dynamic aspects of an information system that supports the business processes in an organization.
- During analysis, behavioral models describe what the internal logic of the processes is without specifying how the processes are to be implemented.
- Later, in the design and implementation phases, the detailed design of the operations contained in the object is fully specified.

Inputs

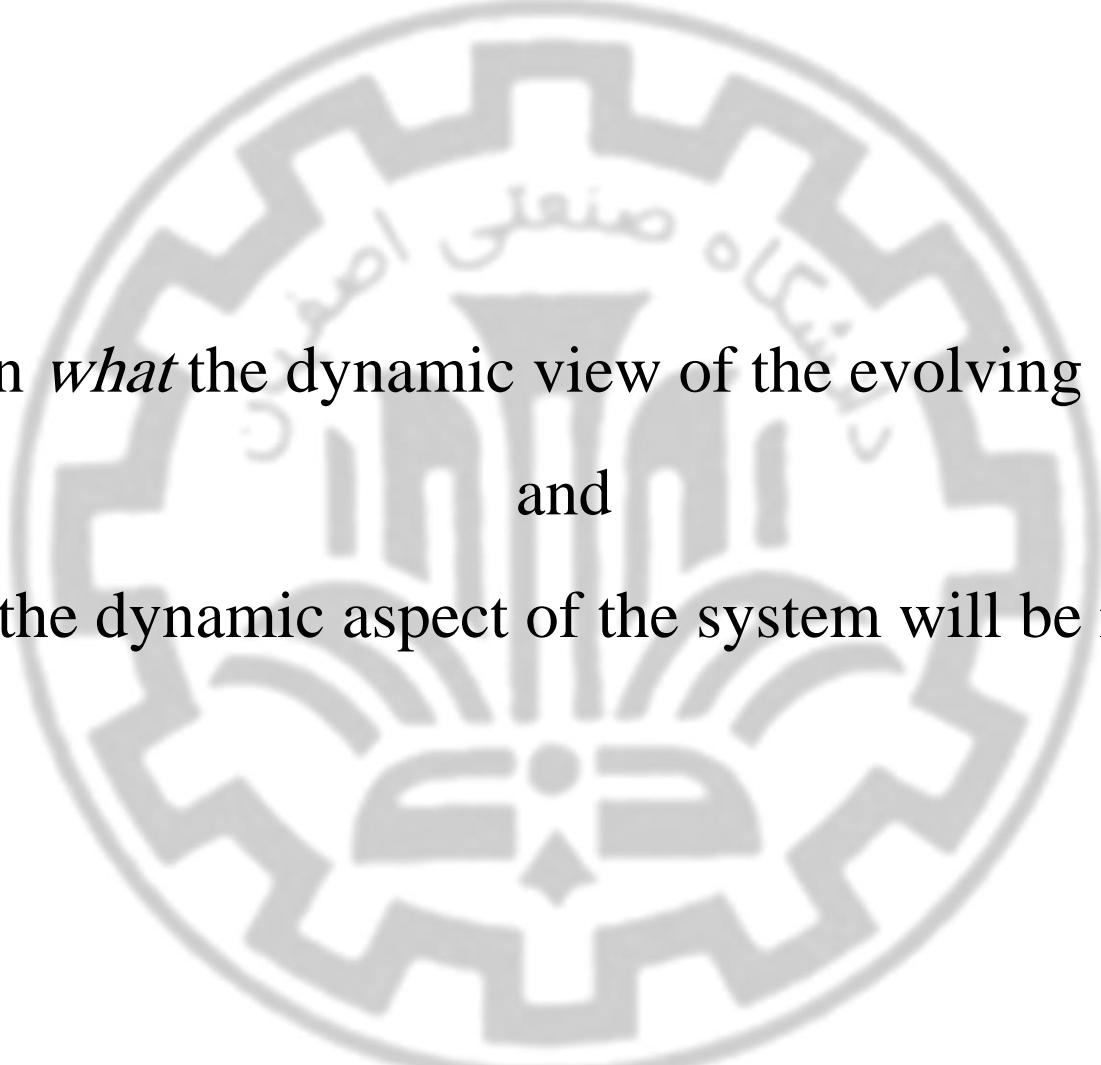
- Use business process and functional models to describe the functional or external behavioral view of an information system.
- Use structural models to depict the internal structural or static view of an information system.

Types of behavioral models

- Behavioral models used to represent the underlying details of a business process portrayed by a use-case model, for example in UML, interaction diagrams (sequence and communication).
 - Interaction diagrams allow the analyst to model the distribution of the behavior of the system over the actors and objects in the system.
- Behavioral model is used to represent the changes that occur in the underlying data, for example in UML, behavioral state machines.

مدلهای رفتاری برای نشان دادن جزئیات زیربنایی یک فرآیند کسبوکار که توسط یک مدل مورد (نمودارهای تعامل (توالی و ارتباطات، UML استفاده به تصویر کشیده میشود، برای مثال در استفاده میشوند.

نمودارهای تعاملی به تحلیلگر اجازه می دهد تا توزیع رفتار سیستم را بر روی پازیگران و اشیاء در سیستم مدل کند.
مدل رفتاری برای نشان دادن تغییراتی که در داده های زیربنایی رخ می دهد استفاده می شود.
ماشین های حالت رفتاری، UML به عنوان مثال در



Focus on *what* the dynamic view of the evolving system is
and
not on *how* the dynamic aspect of the system will be implemented

نشان دادن این است که چگونه اشیاء زیربنایی در یک دامنه مشکل با هم کار می کنند تا یک همکاری برای پشتیبانی از هر یک از موارد استفاده تشکیل دهند.^۰ در حالی که مدل های ساختاری اشیا و روابط بین آنها را نشان می دهند، مدل های رفتاری نمای داخلی فرآیند کسب و کار را که یک مورد استفاده توصیف می کند، به تصویر می کشد.

Primary purposes of behavioral models

- Is to show how the underlying objects in a problem domain will work together to form a collaboration to support each of the use cases.
- Whereas structural models represent the objects and the relationships between them, behavioral models depict the internal view of the business process that a use case describes.
- The process can be shown by the interaction that takes place between the objects that collaborate to support a use case through the use of interaction (sequence and communication) diagrams.
- It is also possible to show the effect that the set of use cases that make up the system has on the objects in the system through the use of behavioral state machines.

فرآیند را می توان با تعاملی که بین اشیایی که برای پشتیبانی از یک مورد استفاده با یکدیگر همکاری می کنند از طریق استفاده از نمودارهای تعامل (توالی و ارتباط) نشان داد.^۰ همچنین می توان با استفاده از ماشین های حالت رفتاری، تأثیری را که مجموعه موارد استفاده تشکیل دهنده سیستم بر روی اشیاء موجود در سیستم می گذارد، نشان داد.

Behavioral modeling

- Is an **iterative process** that iterates not only over the **individual behavioral models** but also over the **functional and structural models**.
- As the behavioral models are created, it is not unusual to **make changes** to the functional and structural models.

یک فرآیند تکراری است که نه تنها بر روی مدل های رفتاری فردی بلکه بر روی مدل های عملکردی و ساختاری نیز تکرار می شود.
با ایجاد مدل های رفتاری، ایجاد تغییرات در مدل های عملکردی و ساختاری
غیر عادی نیست.

Objects, Operations, and Messages

- An object is an instantiation of a *class*, or thing about which we want to capture information.
 - If we were building an appointment system for a doctor's office, classes might include **doctor**, **patient**, and **appointment**. The specific patients is Jim Maloney is considered objects—i.e., *instances* of the patient class.
- Each object has *attributes* that describe information about the object, such as a patient's name.
- Each object also has *behaviors*. At this point in the development of the evolving system, the behaviors are described by *operations*. An operation is nothing more than an action that an object can perform.
 - For example, an appointment object can probably **schedule a new appointment**, **delete an appointment**, and **locate** the next available appointment.

Objects, Operations, and Messages(Cnt'd)

- Each object also can send and receive messages. **Messages** are information sent to objects to tell an object to execute one of its behaviors. Essentially, a message is a function or procedure call from one object to another object.
 - For example, if a patient is new to the doctor's office, the system sends an **insert message** to the application. The patient object receives the instruction (the message) and does what it needs to do to insert the new patient into the system (the behavior).

هر شیء همچنین می تواند پیام ارسال و دریافت کند. پیام ها اطلاعاتی هستند که به اشیا ارسال می شوند تا به یک شیء بگویند یکی از رفتارهای خود را اجرا کند. اساساً یک پیام یک فرآخوانی تابع یا رویه از یک شی به شی دیگر است.

Sequence Diagram

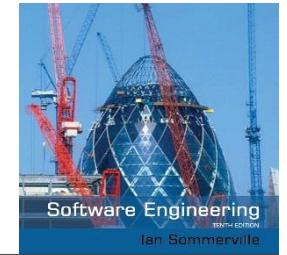
- Is one of two types of interaction diagrams.
- They illustrate the objects that participate in a use case and the messages that pass between them over time for one use case.
- A sequence diagram is a *dynamic model* that shows the explicit sequence of messages that are passed between objects in a defined interaction.
- Because sequence diagrams emphasize the *time-based ordering* of the activity that takes place among a set of objects, they are very helpful for understanding real-time specifications and complex use cases.

آنها اشیایی را که در یک مورد استفاده شرکت می کنند و پیام هایی که در طول زمان بین آنها ارسال می شود را برای یک مورد استفاده نشان می دهد.

نمودار توالی یک مدل پویا است که توالی صریح پیام هایی را نشان می دهد که بین اشیا در یک تعامل تعریف شده ارسال می شود.
از آنجایی که نمودارهای توالی بر ترتیب بر اساس زمان فعالیتی که در میان مجموعه ای از اشیاء انجام می شود تأکید می کنند، برای درک مشخصات بلادرنگ و موارد استفاده پیچیده بسیار مفید هستند.

Sequence diagram(Cnt'd)

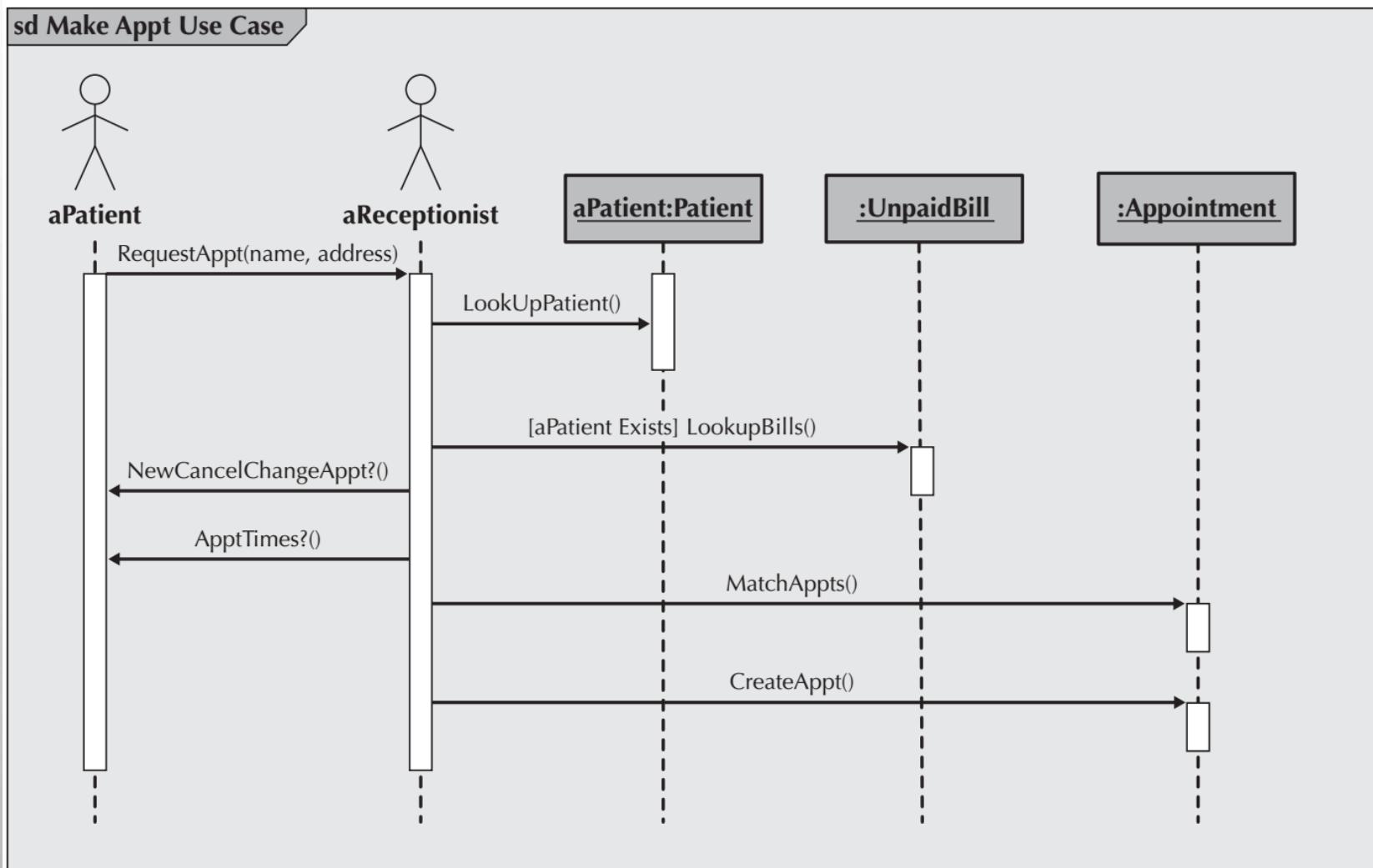
- The sequence diagram can be a *generic sequence diagram* that shows all possible scenarios for a use case, but usually each analyst develops a set of *instance sequence diagrams*, each of which depicts a single scenario within the use case.
- If you are interested in understanding the flow of control of a scenario by time, you should use a sequence diagram to depict this information.
- The diagrams are used throughout the analysis and design phases.
- However, the design diagrams are very implementation specific, often including database objects or specific user interface components as the objects.



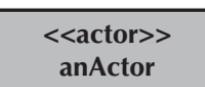
Sequence diagrams

- ✧ Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system.
- ✧ A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
- ✧ The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- ✧ Interactions between objects are indicated by annotated arrows.

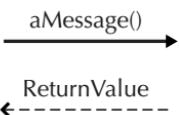
An instance sequence diagram



Elements of a Sequence Diagram

Term and Definition	Symbol
<p>An actor:</p> <ul style="list-style-type: none">■ Is a person or system that derives benefit from and is external to the system.■ Participates in a sequence by sending and/or receiving messages.■ Is placed across the top of the diagram.■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <> in it (alternative).	 anActor 
<p>An object:</p> <ul style="list-style-type: none">■ Participates in a sequence by sending and/or receiving messages.■ Is placed across the top of the diagram.	
<p>A lifeline:</p> <ul style="list-style-type: none">■ Denotes the life of an object during a sequence.■ Contains an X at the point at which the class no longer interacts.	
<p>An execution occurrence:</p> <ul style="list-style-type: none">■ Is a long narrow rectangle placed atop a lifeline.■ Denotes when an object is sending or receiving messages.	

Elements of a sequence diagram(Cnt'd)

A message: <ul style="list-style-type: none">■ Conveys information from one object to another one.■ A operation call is labeled with the message being sent and a solid arrow, whereas a return is labeled with the value being returned and shown as a dashed arrow.	
A guard condition: <ul style="list-style-type: none">■ Represents a test that must be met for the message to be sent.	
For object destruction: <ul style="list-style-type: none">■ An X is placed at the end of an object's lifeline to show that it is going out of existence.	X
A frame: <ul style="list-style-type: none">■ Indicates the context of the sequence diagram.	

بازیگران و اشیایی که در سکانس شرکت می‌کنند با استفاده از نمادهای بازیگر از نمودار مورد استفاده در بالای نمودار قرار می‌گیرند. برای هر یک از اشیاء، نام کلاسی که نمونه‌ای از آن هستند، پس از نام شی داده می‌شود.

Elements of a sequence diagram(Cnt'd)

یک جعبه مستطیلی نازک، به نام رخداد اجرا، روی خط حیاتی قرار داده شده است تا نشان دهد که کلاس‌ها چه زمانی پیام ارسال و دریافت می‌کنند.

- *Actors and objects* that participate in the sequence are placed across the top of the diagram using actor symbols from the use-case diagram. For each of the objects, the **name of the class** of which they are an instance is given **after the object's name**.
- *A dotted line* runs **vertically** below **each actor and object** to denote the *lifeline* of the actors and objects over time. Sometimes an object creates a *temporary object*; in this case, an **X** is placed at the end of the lifeline at the point where the **object is destroyed**. When objects continue to exist in the system after they are used in the sequence diagram, then the lifeline continues to the bottom of the diagram.
- A thin rectangular box, called the *execution occurrence*, is overlaid onto the lifeline to show when the classes are sending and receiving messages.

پیام یک ارتباط بین اشیاء است که اطلاعات را با این انتظار که فعالیتی در پی خواهد داشت منتقل می کند.

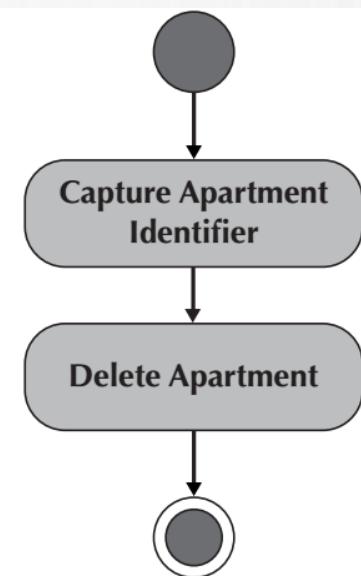
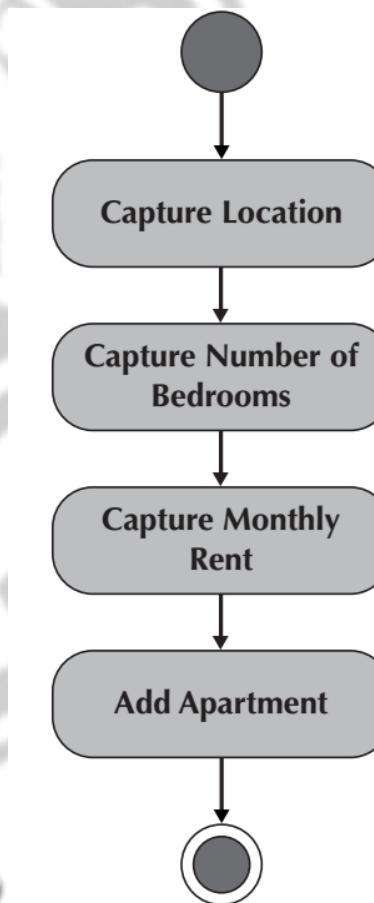
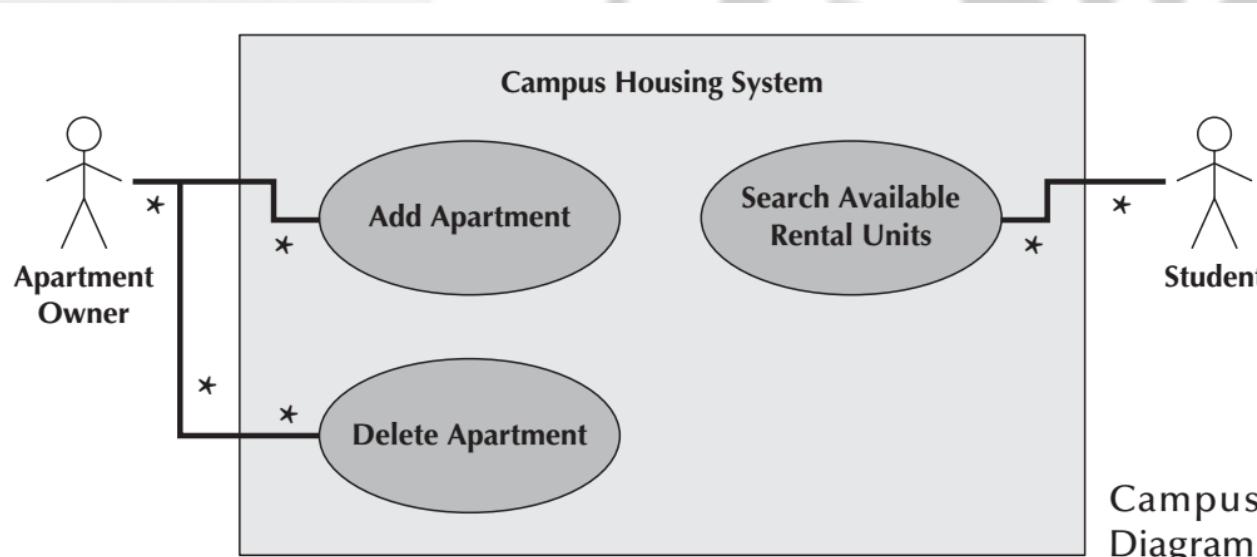
Elements of a sequence diagram(Cnt'd)

- A *message* is a communication between objects that conveys information with the expectation that activity will ensue.
- Two types of messages: operation call and return. *Operation call* messages passed between objects are shown using solid lines connecting two objects with an arrow on the line showing which way the message is being passed. Argument values for the message are placed in parentheses next to the message's name. The order of messages goes from the top to the bottom of the page, so messages located higher on the diagram represent messages that occur earlier on in the sequence, versus the lower messages that occur later. A *return message* is depicted as a dashed line with an arrow on the end of the line portraying the direction of the return. The information being returned is used to label the arrow.

Elements of a sequence diagram(Cnt'd)

- At times a message is sent only if a *condition* is met. In those cases, the *condition* is placed between a set of brackets, []. The condition is placed in front of the message name.
- An object can send a message to itself. This is known as *self-delegation*.

Campus Housing Service Functional Models



Campus
Diagram

Campus Housing Service Functional Models(Cnt'd)

Use Case Name:	Add Apartment	ID:	1	Importance Level:	High				
Primary Actor:	Apartment Owner		Use Case Type:	Detail, Essential					
Stakeholders and Interests: Apartment Owner – wants to advertise available apartment Campus Housing Service – provides a service that enables the apartment owners to rent their available apartments									
Brief Description: This use case describes how the campus housing service can maintain an up-to-date listing of available apartments.									
Trigger: Apartment Owner wants to add an available apartment									
Type: External									
Relationships: Association: Apartment Owner Include: Extend: Generalization:									
Normal Flow of Events: 1. Capture the location of the apartment. 2. Capture the number of bedrooms in the apartment. 3. Capture the monthly rent of the apartment. 4. Add the apartment to the listing of available apartments.									
SubFlows:									
Alternate/Exceptional Flows:									

Use Case Name:	Delete Apartment	ID:	2	Importance Level:	High				
Primary Actor:	Apartment Owner		Use Case Type:	Detail, Essential					
Stakeholders and Interests: Apartment Owner – wants to delist apartment Campus Housing Service – provides a service that enables the apartment owners to rent their available apartments									
Brief Description: This use case describes how the campus housing service can maintain an up-to-date listing of available apartments.									
Trigger: Apartment Owner wants to delete an available apartment									
Type: External									
Relationships: Association: Apartment Owner Include: Extend: Generalization:									
Normal Flow of Events: 1. Capture the apartment identifier. 2. Delete the apartment from the listing of available apartments.									
SubFlows:									
Alternate/Exceptional Flows:									

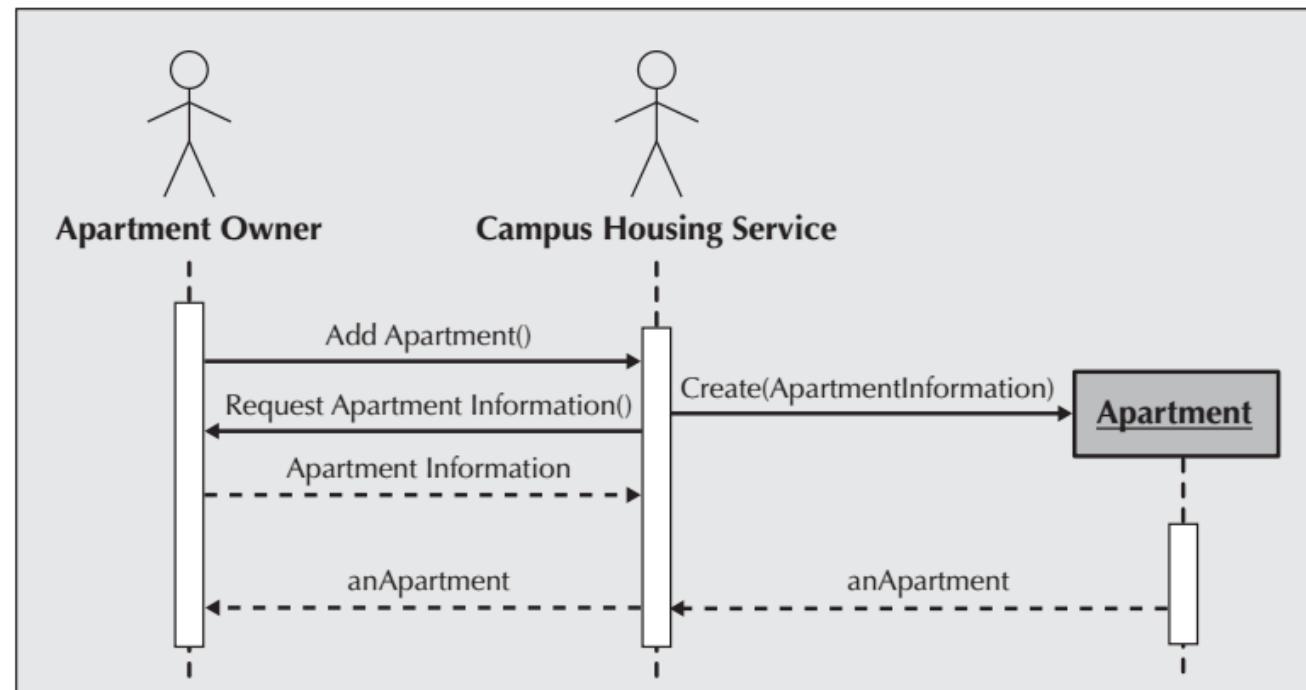
Campus Housing Service

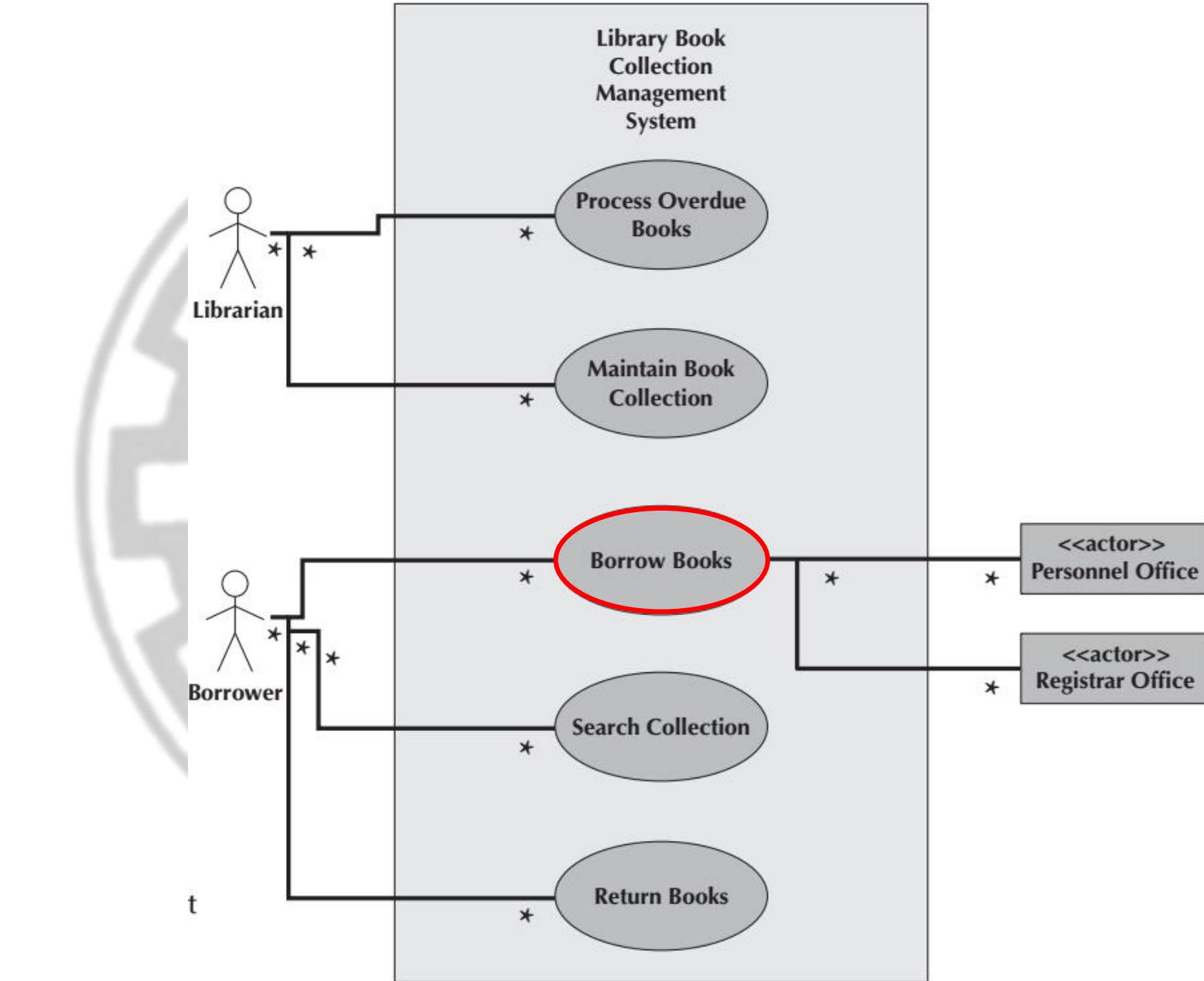
structural Models

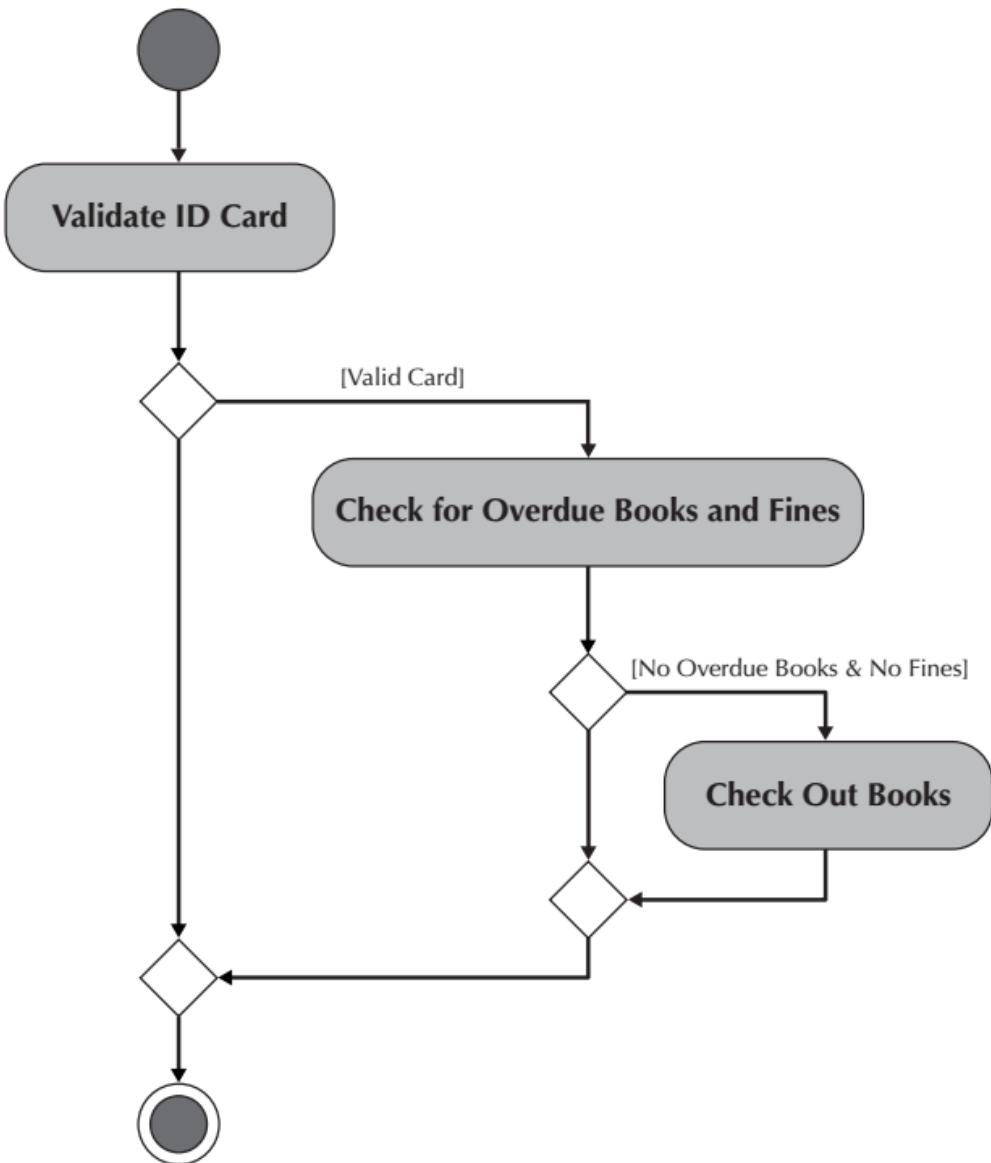


Front:		
Class Name: Apartment Owner	ID: 1	Type: Concrete, Domain
Description: An apartment owner who has apartments for rent		Associated Use Cases: 2
<p>Responsibilities</p> <p>Add Apartment</p> <hr/> <p>Delete Apartment</p> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>		<p>Collaborators</p> <p>Apartment</p> <hr/> <p>Apartment</p> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>
Back:		
<p>Attributes:</p> <p>Name (string)</p> <hr/> <p>Address (address)</p> <hr/> <p>Phone number (phone number)</p> <hr/> <p>Email (Email address)</p> <hr/>		
<p>Relationships:</p> <p>Generalization (a-kind-of):</p> <hr/> <p>Aggregation (has-parts):</p> <hr/> <p>Other Associations:</p> <p>Apartment</p> <hr/> <hr/>		

Sequence Diagram for the Add Apartment Use Case







Overview Description for the Borrow Books Use Case

Use Case Name: Borrow Books	ID: <u>2</u>	Importance Level: <u>High</u>		
Primary Actor: Borrower	Use Case Type: Detail, Essential			
<p>Stakeholders and Interests:</p> <p>Borrower—wants to check out books</p> <p>Librarian—wants to ensure borrower only gets books deserved</p>				
<p>Brief Description: This use case describes how books are checked out of the library.</p>				
<p>Trigger: Borrower brings books to check out desk.</p>				
<p>Type: External</p>				
<p>Relationships:</p> <p>Association: Borrower, Personnel Office, Registrar's Office</p> <p>Include:</p> <p>Extend:</p> <p>Generalization:</p>				

Flow Descriptions for the Borrow Books Use Case

Normal Flow of Events:

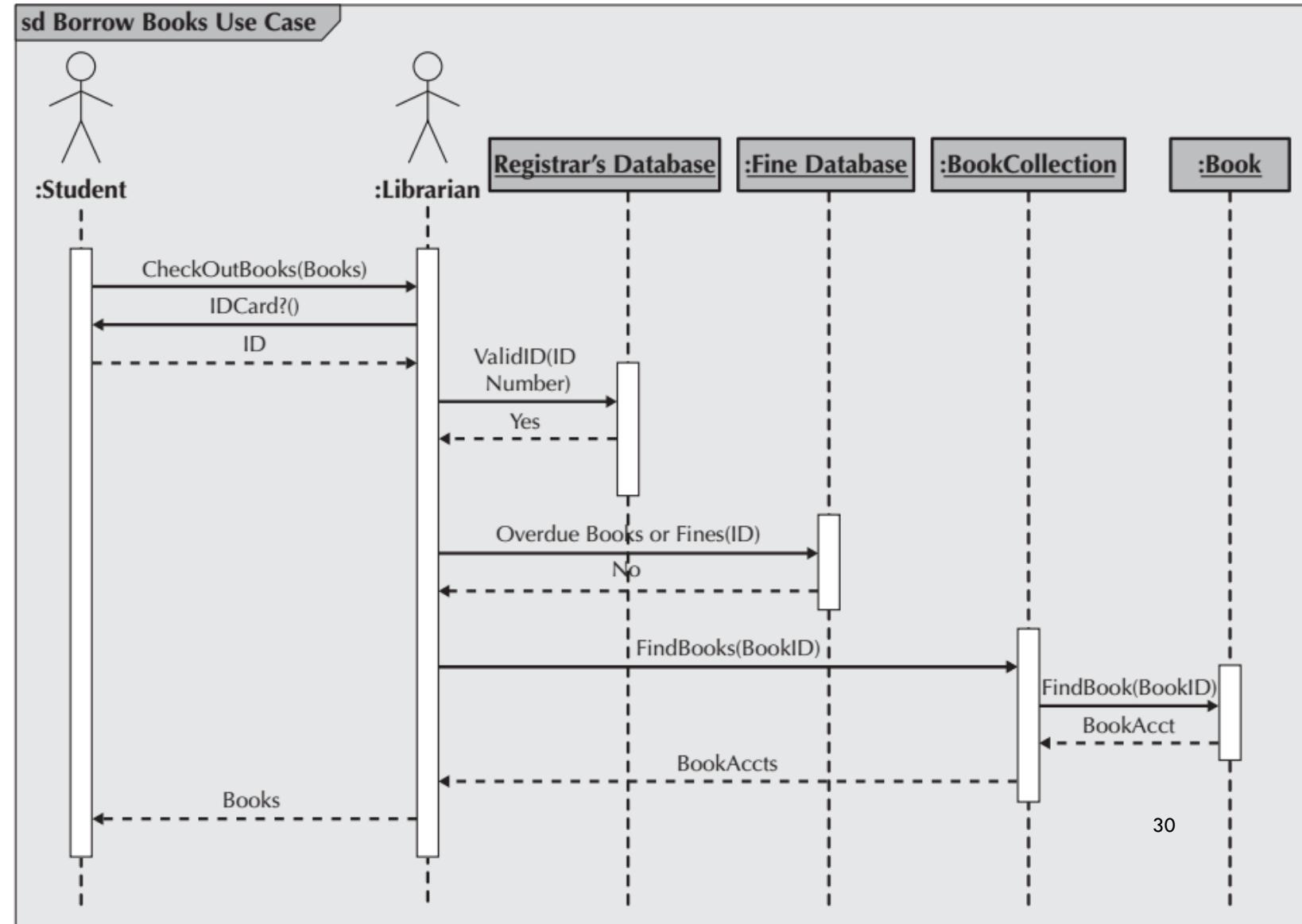
1. The Borrower brings books to the Librarian at the check out desk.
2. The Borrower provides Librarian their ID card.
3. The Librarian checks the validity of the ID Card.
 - If the Borrower is a Student Borrower, Validate ID Card against Registrar's Database.
 - If the Borrower is a Faculty/Staff Borrower, Validate ID Card against Personnel Database.
 - If the Borrower is a Guest Borrower, Validate ID Card against Library's Guest Database.
4. The Librarian checks whether the Borrower has any overdue books and/or fines
5. The Borrower checks out the books

SubFlows:

Alternate/Exceptional Flows:

- 4a The ID Card is invalid, the book request is rejected.
- 5a The Borrower either has overdue books, fines, or both, the book request is rejected.

Sequence Diagram of the Borrow Books Use Case for Students with a Valid ID and No Overdue Books or Fines



Reference

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

Software Engineering I

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

2021

Chapter 6

Behavioral modeling(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

Behavioral model

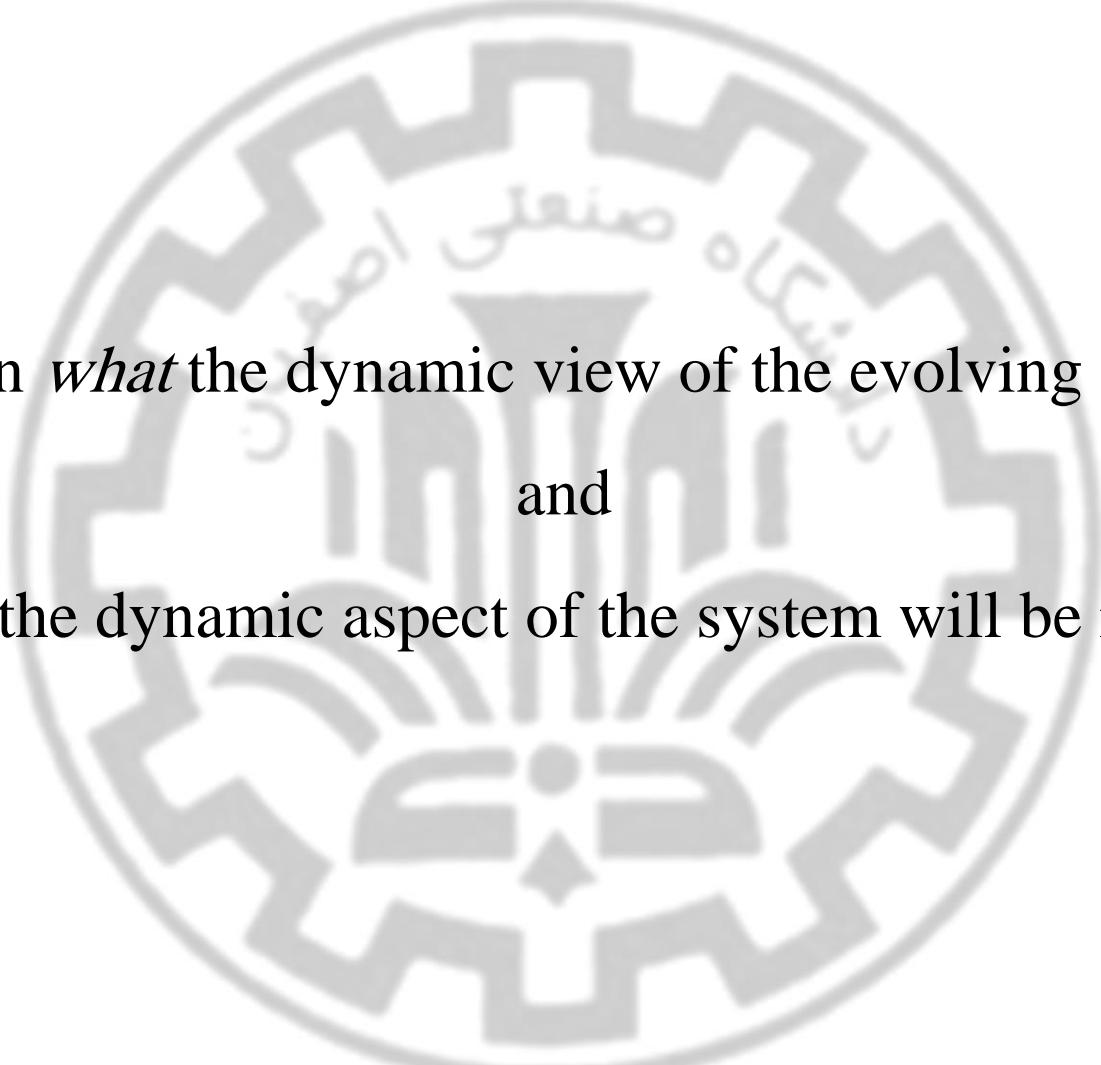
- Describe the internal dynamic aspects of an information system that supports the business processes in an organization.
- During analysis, behavioral models describe what the internal logic of the processes is without specifying how the processes are to be implemented.
- Later, in the design and implementation phases, the detailed design of the operations contained in the object is fully specified.

Inputs

- Use business process and functional models to describe the functional or external behavioral view of an information system.
- Use structural models to depict the internal structural or static view of an information system.

Types of behavioral models

- Behavioral models used to represent the underlying details of a business process portrayed by a use-case model, for example in UML, **interaction** diagrams (sequence and communication).
 - Interaction diagrams allow the analyst to model the distribution of the behavior of the system over the actors and objects in the system.
- Behavioral model is used to represent the **changes** that occur in the underlying data, for example in UML, behavioral state machines.



Focus on *what* the dynamic view of the evolving system is
and
not on *how* the dynamic aspect of the system will be implemented

- نمودارهای ارتباطی، مانند نمودارهای توالی، اساساً نمایی از جنبه های دینامیکی یک سیستم شی گرا را • ارائه می دهند
- نشان می دهد که چگونه اعضای مجموعه ای از اشیاء برای اجرای یک مورد استفاده یا یک سناریوی • مورد استفاده با یکدیگر همکاری می کنند

Communication Diagrams

- Communication diagrams, like sequence diagrams, essentially provide a view of the dynamic aspects of an object-oriented system.
- Show how the members of a set of objects collaborate to implement a use case or a use-case scenario.
- Used to model all the interactions among a set of collaborating objects.
- A communication diagram can portray how dependent the different objects are on one another.
- A communication diagram is essentially an object diagram that shows message-passing relationships instead of aggregation or generalization associations.

- برای مدل سازی تمام تعاملات بین مجموعه ای از اشیاء همکار استفاده می شود
 - یک نمودار ارتباطی می تواند نشان دهد که اشیاء مختلف چقدر به یکدیگر وابسته هستند
- نمودار ارتباطی اساساً یک نمودار شی است که به جای پیوندهای تجمعی یا تعمیم، روابط انتقال پیام را نشان می دهد

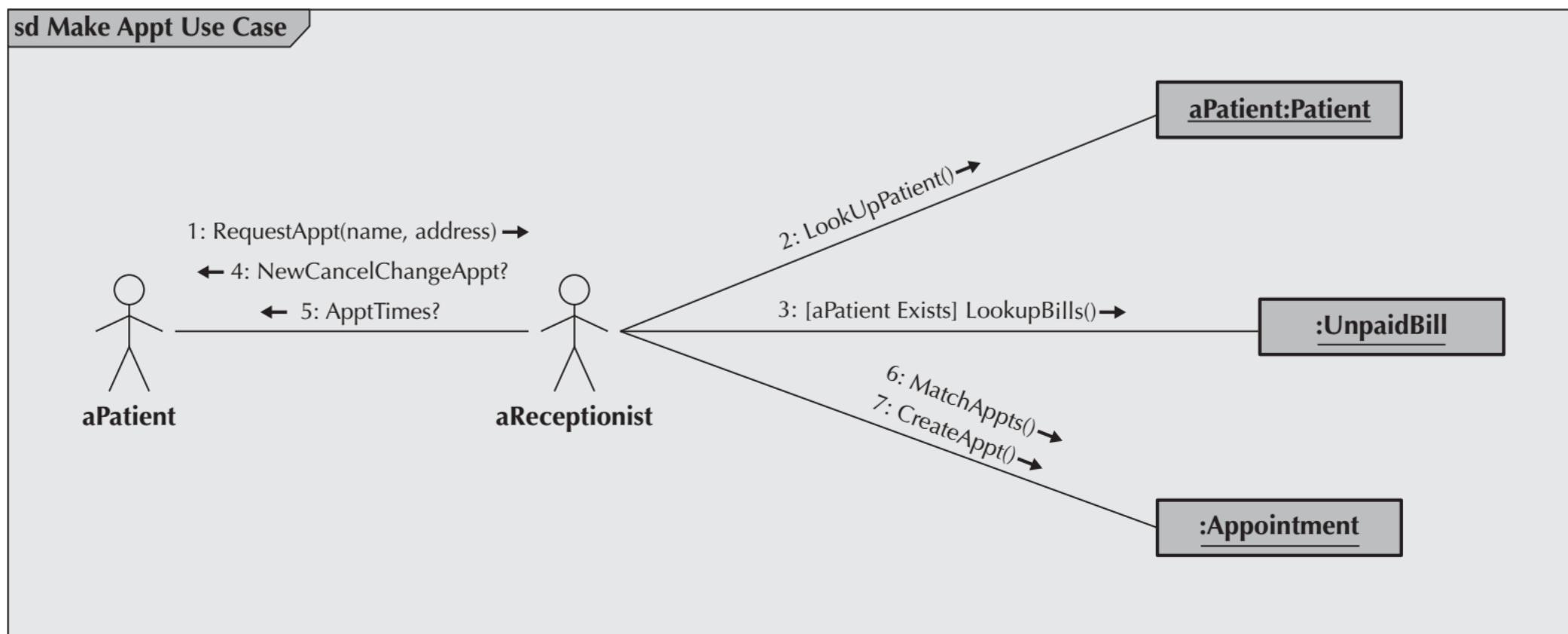
نمودارهای ارتباطی بر جریان پیام ها از طریق مجموعه ای از اشیاء تأکید دارند، در حالی که نمودارهای دنباله ای بر ترتیب زمانی پیام های ارسال شده تمرکز دارند.

Communication Dia. vs. Sequence Dia.

- Communication diagrams emphasize the flow of messages through a set of objects, whereas the sequence diagrams focus on the time ordering of the messages being passed.
- To understand the flow of control over a set of collaborating objects or to understand which objects collaborate to support business processes, a communication diagram can be used.
- For time ordering of the messages, a sequence diagram should be used.
- In some cases, both can be used to more fully understand the dynamic activity of the system.

برای درک جریان کنترل روی مجموعه ای از اشیاء همکار یا درک اینکه •
کدام اشیاء برای پشتیبانی از فرآیندهای تجاری با یکدیگر همکاری می کنند
• می توان از یک نمودار ارتباطی استفاده کرد
برای ترتیب زمانی پیام ها باید از نمودار توالی استفاده شود

An example



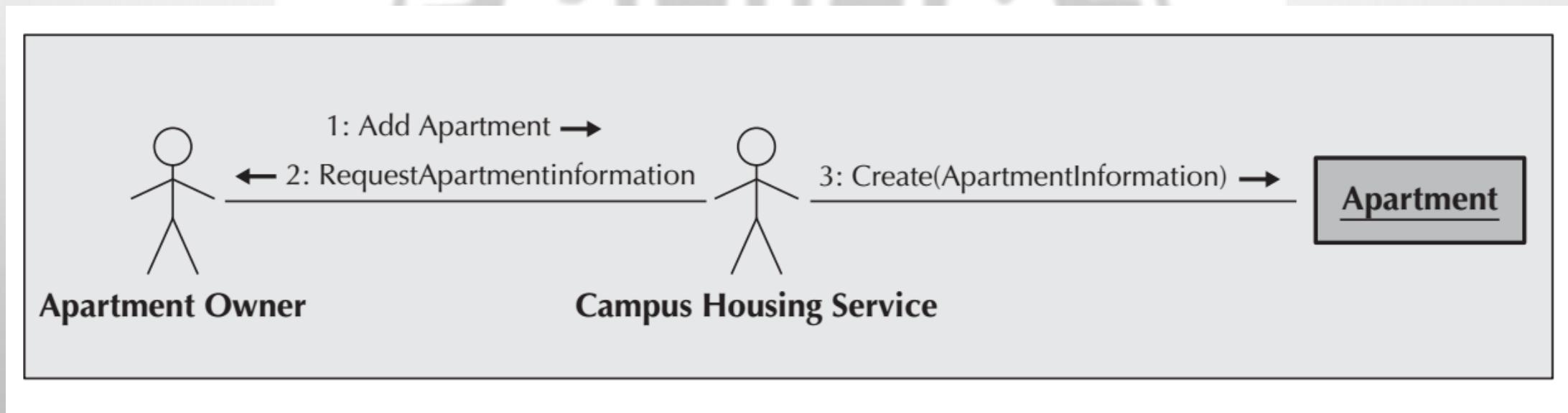
Elements of a Communication Diagram

Term and Definition	Symbol
<p>An actor:</p> <ul style="list-style-type: none"> ■ Is a person or system that derives benefit from and is external to the system. ■ Participates in a collaboration by sending and/or receiving messages. ■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <> in it (alternative). 	 anActor <div style="border: 1px solid black; padding: 5px; display: inline-block;"><> anActor</div>
<p>An object:</p> <ul style="list-style-type: none"> ■ Participates in a collaboration by sending and/or receiving messages. 	<div style="border: 1px solid black; padding: 5px; display: inline-block;">anObject : aClass</div>
<p>An association:</p> <ul style="list-style-type: none"> ■ Shows an association between actors and/or objects. ■ Is used to send messages. 	<hr/>
<p>A message:</p> <ul style="list-style-type: none"> ■ Conveys information from one object to another one. ■ Has direction shown using an arrowhead. ■ Has sequence shown by a sequence number. 	<div style="border-bottom: 1px solid black; padding-bottom: 5px; display: inline-block;">SeqNumber: aMessage →</div>
<p>A guard condition:</p> <ul style="list-style-type: none"> ■ Represents a test that must be met for the message to be sent. 	<div style="border-bottom: 1px solid black; padding-bottom: 5px; display: inline-block;">SeqNumber: [aGuardCondition]: aMessage →</div>
<p>A frame:</p> <ul style="list-style-type: none"> ■ Indicates the context of the communication diagram. 	<div style="border: 1px solid black; padding: 5px; display: inline-block;">Context</div>

Difference between Communication Dia. And Sequence Dia.

- Unlike the sequence diagram, the communication diagram **does not have a means to explicitly show an object being deleted or created**. It is assumed that when a delete, destroy, or remove message is sent to an object, it will go out of existence, and a create or new message will cause a new object to come into existence.
- Another difference between the two interaction diagrams is that the communication diagram **never shows returns from message sends**, whereas the sequence diagram can optionally show them.

Communication Diagram for the Add Apartment Use Case



برخی از کلاسها در نمودارهای کلاس، مجموعهای از اشیاء را نشان میدهند که کاملاً پویا هستند، زیرا در طول مسیر خود از حالت‌های مختلفی عبور می‌کنند. وجود داشتن ماشین حالت رفتاری یک مدل پویا است که حالت‌های مختلفی را که یک شی منفرد در طول عمر خود در واکنش به رویدادها از آن عبور می‌کند، همراه با پاسخ‌ها و اعمال خود نشان می‌دهد.

Behavioral State Machines

ماشین حالت رفتاری حالت‌های مختلف شی را نشان می‌دهد و اینکه چه اتفاقاتی باعث تغییر جسم از حالتی به حالتی دیگر می‌شود.

- Some of the classes in the *class diagrams* represent a set of objects that are quite dynamic in that they pass through a variety of states over the course of their existence.
- A behavioral state machine is a dynamic model that shows the different states through which a single object passes during its life in response to events, along with its responses and actions.
- Typically, behavioral state machines are not used for all objects; rather, behavioral state machines are used with complex objects to further define them and to help simplify the design of algorithms for their methods.
- The behavioral state machine shows the different states of the object and what events cause the object to change from one state to another.
- Behavioral state machines should be used to help understand the dynamic aspects of a single class and how its instances evolve over time, Unlike interaction diagrams that show how a particular use case or use-case scenario is executed over a set of classes.

ماشینهای حالت رفتاری باید برای کمک به درک جنبه‌های دینامیکی یک کلاس استفاده شوند

وضعیت یک شی با ارزش ویژگی های آن و روابط آن با اشیاء دیگر در یک نقطه خاص از زمان تعریف می شود. • صفات یا خصوصیات یک شیء بر وضعیتی که در آن قرار دارد تأثیر می گذارد.

State

- The *state* of an object is defined by the **value of its attributes** and **its relationships with other objects** at a particular **point in time**.
- The **attributes or properties** of an object affect the state that it is in;
- Not all attributes or attribute changes will make a difference.
- Is a **set of values** that describes an object at a specific point in time and represents a point in an object's life in which **it satisfies some condition**, **performs some action**, or **waits for something to happen**.

• همه خصیصه ها یا تغییرات صفت تفاوتی ایجاد نمی کنند.

• مجموعه ای از مقادیر است که یک شی را در یک نقطه خاص از زمان توصیف می کند و نقطه ای از زندگی یک شی را نشان می دهد که در آن شرایطی را براورده می کند، عملی را انجام می دهد یا منتظر اتفاقی است.

Event

- An *event* is something that takes place at a certain point in time and changes a value or values that describe an object, which, in turn, changes the object's state.
- It can be a designated condition becoming true, the receipt of the call for a method by an object, or the passage of a designated period of time.

رویداد چیزی است که در یک نقطه خاص از زمان رخ می دهد و مقدار یا مقادیری را تغییر می دهد که یک شی را توصیف می کند، که به نوبه خود حالت شی را تغییر می دهد.
این می تواند یک شرط تعیین شده واقعی شود، دریافت فراخوانی برای یک متود توسط یک شی •
یا گذر از یک دوره زمانی تعیین شده باشد

Transition

- A *transition* is a relationship that represents the movement of an object from one state to another state.
- Some transitions have a guard condition. A *guard condition* is a Boolean expression that includes attribute values, which allows a transition to occur only if the condition is true.
- An object typically moves from one state to another based on the outcome of an action triggered by an event.

انتقال رابطه‌ای است که نشان دهنده حرکت یک شی از یک حالت به حالت دیگر است.

برخی از ترانزیشن‌ها شرایط نگهبانی دارند.

شرط نگهبان یک عبارت پولی است که شامل مقادیر مشخصه می‌شود، که تنها در صورت

درست بودن شرط اجازه میدهد که یک انتقال رخ دهد.

یک شی به طور معمول از یک حالت به حالت دیگر بر اساس نتیجه یک عمل تحریک شده.

توسط یک رویداد حرکت می‌کند.

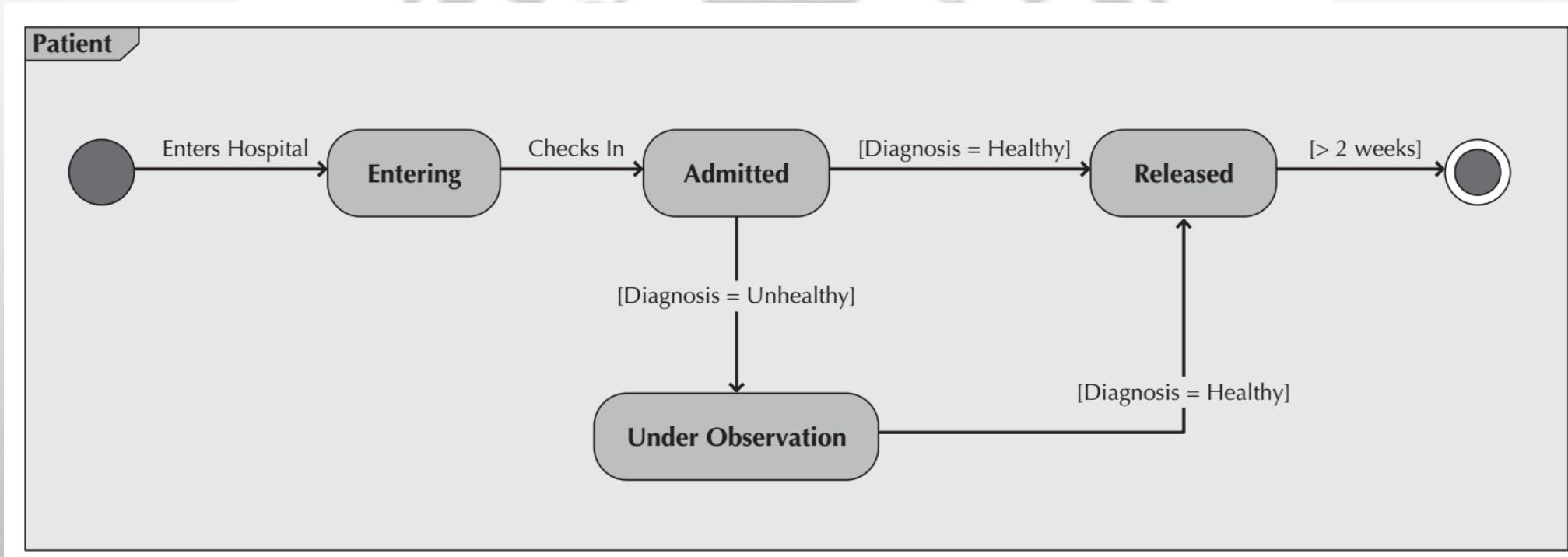
Action and Activity

- An *action* is an atomic, non-decomposable process that cannot be interrupted. From a practical perspective, actions take zero time, and they are associated with a transition.
- In contrast, an *activity* is a non-atomic, decomposable process that can be interrupted. Activities take a long period of time to complete, and they can be started and stopped by an action.

عمل یک فرآیند اتمی و غیرقابل تجزیه است که نمی توان آن را قطع کرد. از منظر عملی اقدامات زمان صفر دارند و با یک انتقال همراه هستند.

در مقابل، یک فعالیت یک فرآیند غیر اتمی و تجزیه پذیر است که می تواند قطع شود. انجام فعالیت ها زمان زیادی می برد و می توان آنها را با یک اقدام شروع کرد و متوقف کرد.

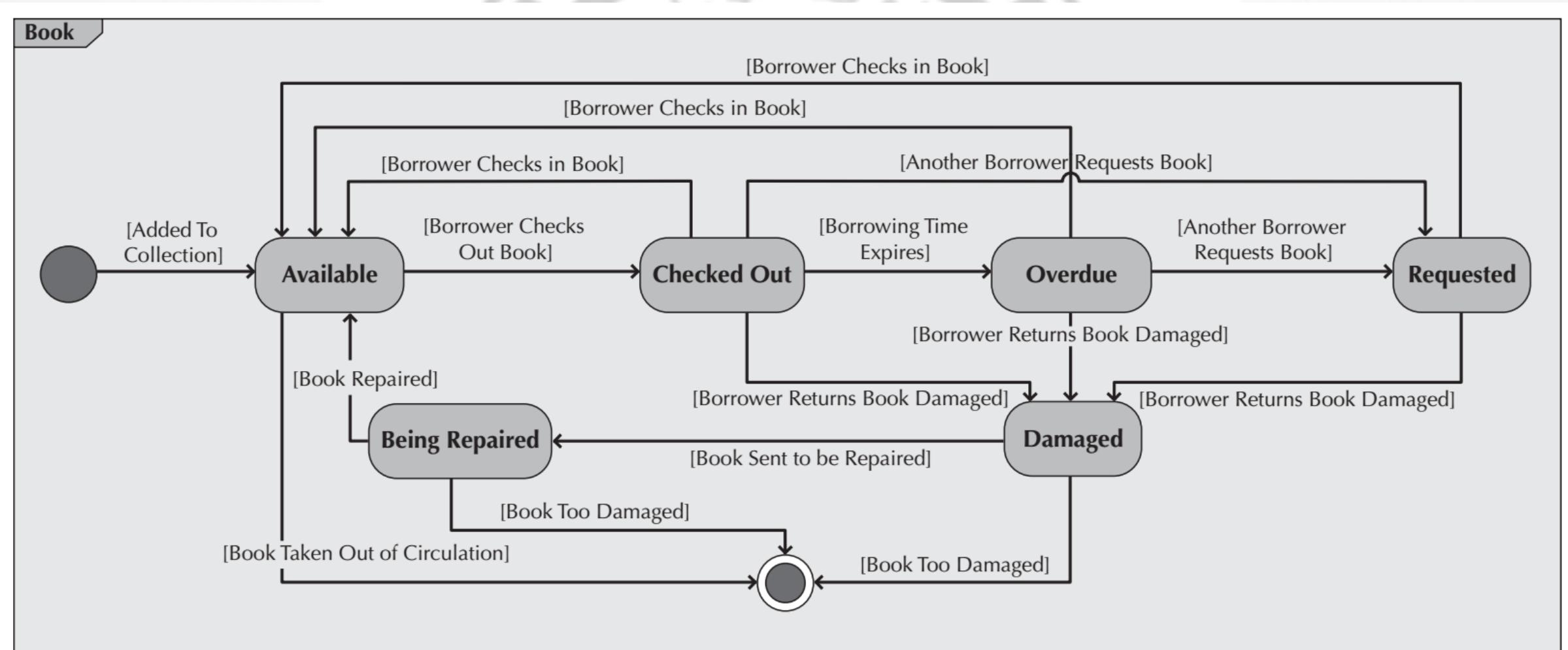
An example



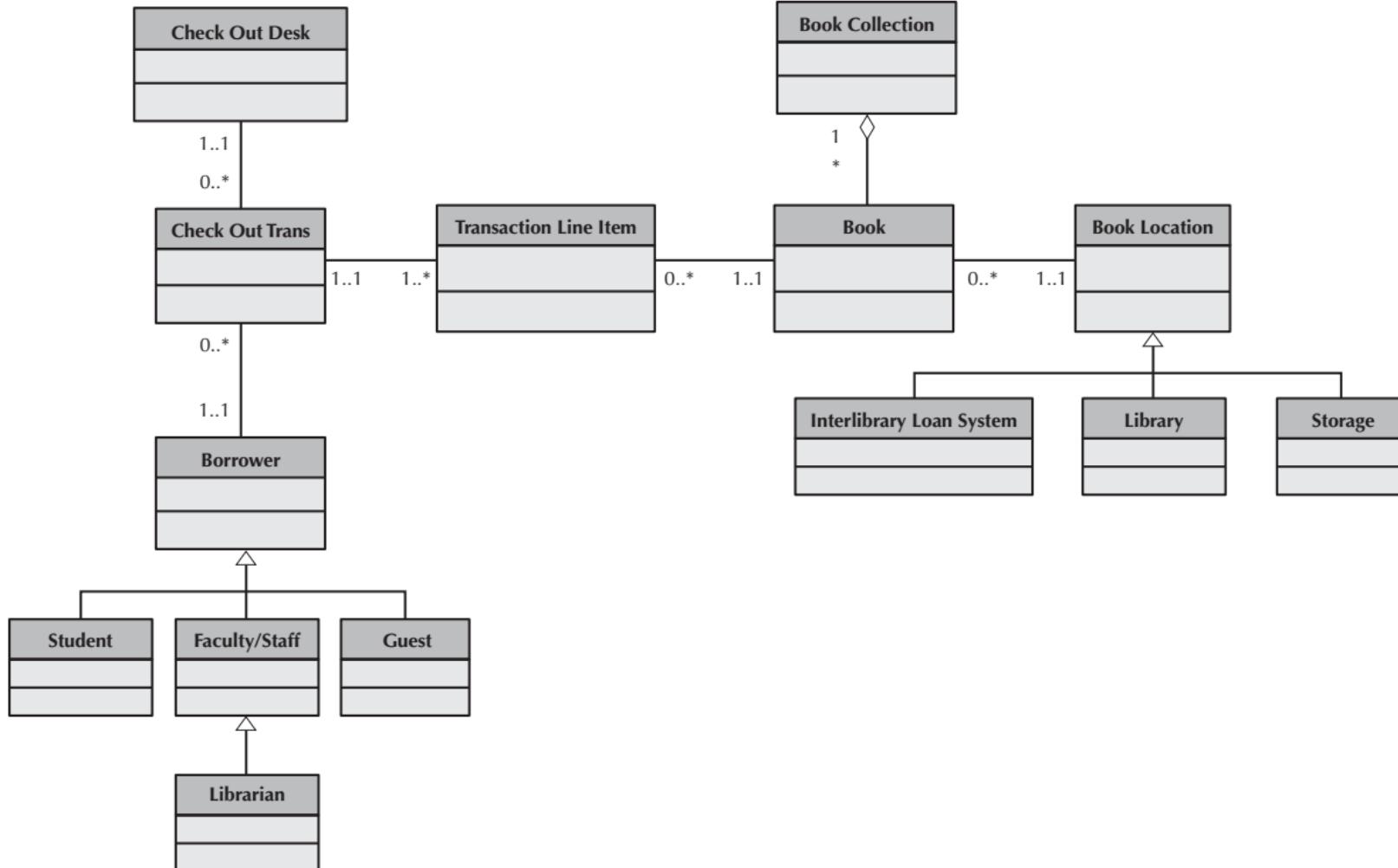
Elements of a Behavioral State Machine

Term and Definition	Symbol
<p>A state:</p> <ul style="list-style-type: none">■ Is shown as a rectangle with rounded corners.■ Has a name that represents the state of an object.	 aState
<p>An initial state:</p> <ul style="list-style-type: none">■ Is shown as a small, filled-in circle.■ Represents the point at which an object begins to exist.	
<p>A final state:</p> <ul style="list-style-type: none">■ Is shown as a circle surrounding a small, filled-in circle (bull's-eye).■ Represents the completion of activity.	
<p>An event:</p> <ul style="list-style-type: none">■ Is a noteworthy occurrence that triggers a change in state.■ Can be a designated condition becoming true, the receipt of an explicit signal from one object to another, or the passage of a designated period of time.■ Is used to label a transition.	anEvent
<p>A transition:</p> <ul style="list-style-type: none">■ Indicates that an object in the first state will enter the second state.■ Is triggered by the occurrence of the event labeling the transition.■ Is shown as a solid arrow from one state to another, labeled by the event name.	
<p>A frame:</p> <ul style="list-style-type: none">■ Indicates the context of the behavioral state machine.	Context

Behavioral State Machine for an Instance of the Book Class in the Library Book Collection Management System



Class Diagram for the Library Book Collection Management System



Verifying and Validating The Behavioral Models

- First, every actor and object included on a sequence diagram must be included as an actor and an object on a communication diagram, and vice versa.
- Second, every message that is included on a sequence diagram must appear as a message on an association in the corresponding communication diagram, and vice versa.

Verifying and Validating The Behavioral Model(Cnt'd)

- Third, if a guard condition appears on a message in the sequence diagram, there must be an equivalent guard condition on the corresponding communication diagram, and vice versa.
- Fourth, the sequence number included as part of a message label in a communications diagram implies the sequential order in which the message will be sent. Therefore, it must correspond to the top-down ordering of the messages being sent on the sequence diagram.
- Fifth, all transitions contained in a behavior state machine must be associated with a message being sent on a sequence and communication diagram.

Because object-oriented development is *iterative* and *incremental*,
continuous modification of the evolving models (functional, structural, and behavioral)
of the system is to be expected.

What should you do for your project?

1. Create behavioral models.

We will work in the lab.

Reference

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