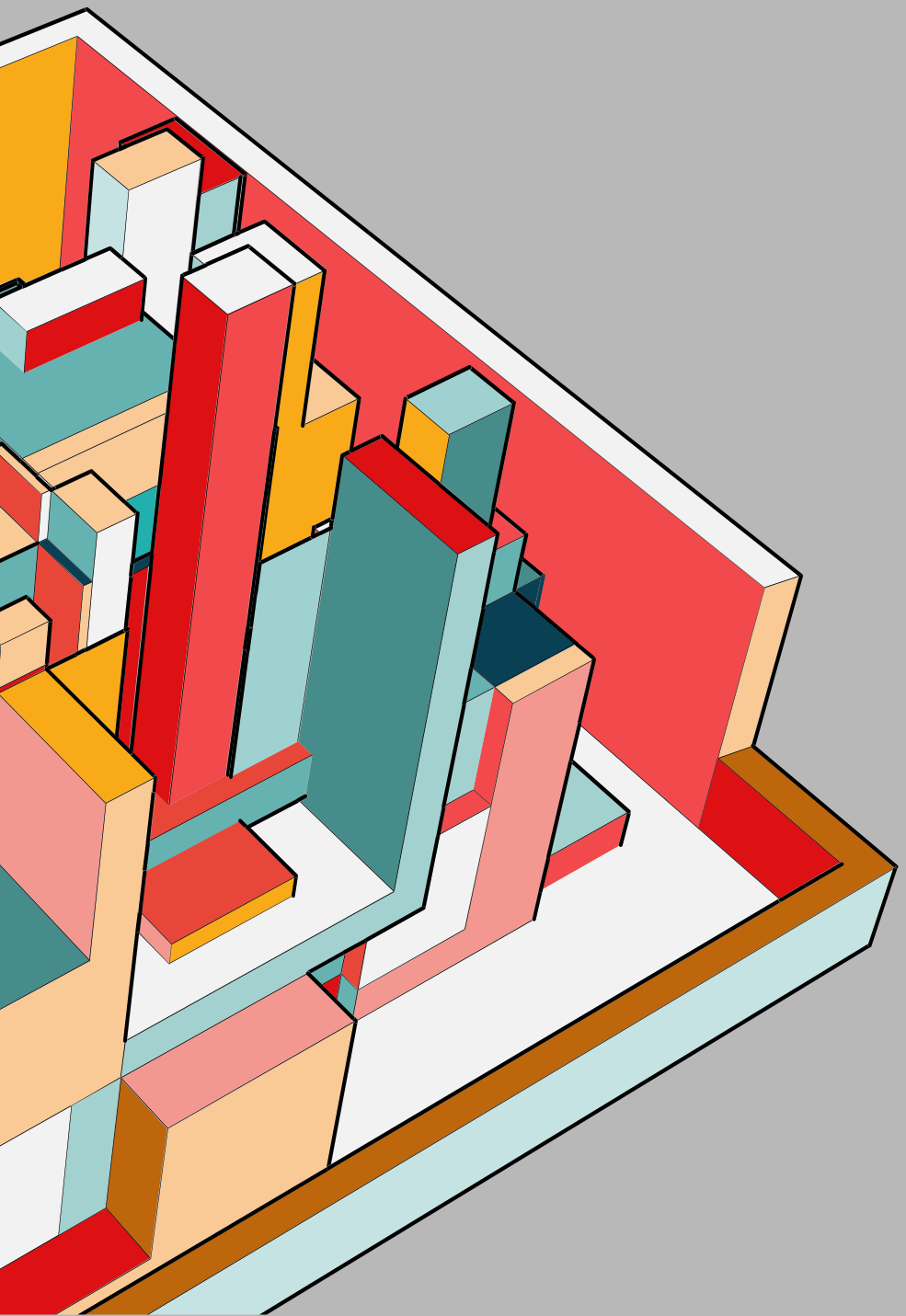


# تصميم وهيكلة البرمجيات
# ITSE411

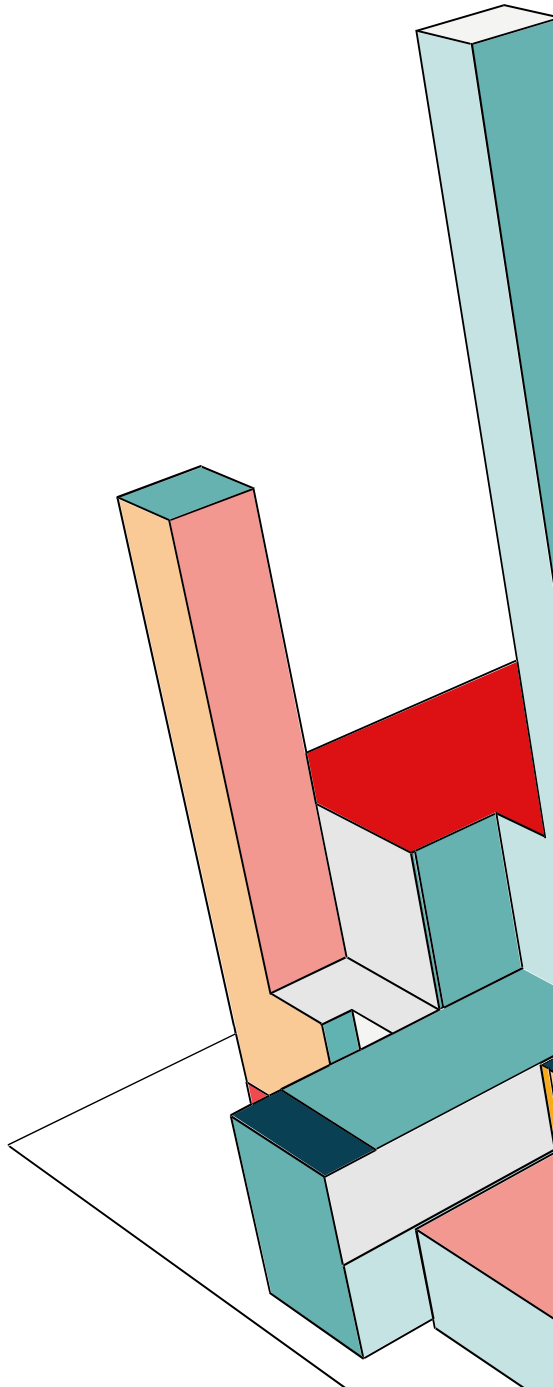
المحاضرة الثانية

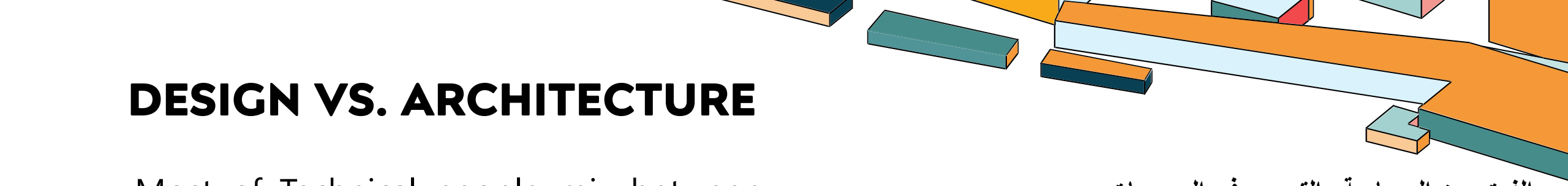
# التصميم والهيكلة

بيرم زرتي - 2025

# محاور المحاضرة

- ما بين التصميم والهيكلة. Design vs. Architecture
- **أنواع أدوار المصمم والمعماري في هندسة البرمجيات** Design and Architecture Roles
- أنواع أدوار المصمم والمعماري في هندسة البرمجيات
- أهمية تحليل وجمع المتطلبات واثرها على عملية التصميم والهيكلة





# DESIGN VS. ARCHITECTURE

- Most of Technical people mix between architecture and design.

Architecture (IEEE definition): is the fundamental organization of a software system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.

Software architecture: is the high-level structure or blueprint of a software system. It defines how different components of a system interact with each other, the overall organization of the system,

**الفرق بين المعمارية والتصميم في البرمجيات**

يخلط الكثير من المختصين التقنيين بين مفهومي **المعمارية البرمجية** و**التصميم البرمجي**، رغم أن لكل منهما دورًا مختلفًا في دورة حياة تطوير البرمجيات.

**المعمارية البرمجية** Software Architecture

وفقًا لتعريف IEEE "المعمارية هي التنظيم الأساسي للنظام البرمجي، كما يتجلى في مكوناته، وعلاقاتها ببعضها البعض وبالبيئة المحيطة، والمبادئ التي توجه تصميمه وتطوره."

بمعنى آخر، المعمارية هي **المخطط البنيوي عالي المستوى** للنظام، وتشمل:

- تحديد المكونات الأساسية للنظام
- كيفية تفاعل هذه المكونات مع بعضها البعض
- العلاقة بين النظام والبيئة الخارجية (مثل قواعد البيانات، الشبكات، المستخدمين)
- المبادئ التي تحكم التطوير المستقبلي للنظام

**التصميم البرمجي** Software Design

أما التصميم، فهو أكثر **تفصيلًا وتقنيًا**، ويُعنى بكيفية تنفيذ كل مكون من مكونات المعمارية، ويشمل:

- ❑ اختيار الخوارزميات
- ❑ تحديد البنية الداخلية للفئات Classes والوحدات Modules
- ❑ تنظيم البيانات وتدفقها داخل النظام
- ❑ كتابة الكود وتحديد واجهات الاستخدام

# DESIGN VS. ARCHITECTURE

Software Design:

Software design is a lower-level activity compared to architecture, concerned with the detailed specifications of the components that make up a system. It focuses on how the system will be implemented

software design is a process of building a program while satisfying a program's functional requirements and not violating it's nonfunctional constraints.

Detail design  is the process of dealing with individual component. Particularly, with respect to their data structure and their algorithms.

**تصميم البرمجيات** Software Design

**تصميم البرمجيات** هو نشاط ذو مستوى أدنى مقارنةً بالمعمارية البرمجية، ويُعنى بوضع **المواصفات التفصيلية** للمكونات التي يتكوّن منها النظام. يركز التصميم على **كيفية تنفيذ النظام**، وليس فقط على ما يجب أن يفعله.

**تعريف التصميم البرمجي**

"تصميم البرمجيات هو عملية بناء برنامج يحقق المتطلبات الوظيفية، دون أن ينتهك القيود غير الوظيفية."

بمعنى آخر، يهدف التصميم إلى تحويل المتطلبات إلى **خطة تنفيذية واضحة** تشمل البنية الداخلية للمكونات، وتفاصيل البيانات والخوارزميات.

**مستويات التصميم**

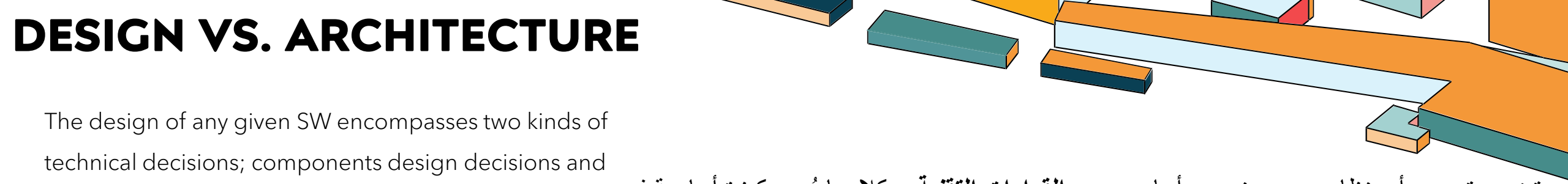
**التصميم العام** High-Level Design

- يحدد كيفية تقسيم النظام إلى وحدات أو مكونات رئيسية.
- يوضح العلاقات بين هذه المكونات وواجهات التفاعل بينها.

**التصميم التفصيلي** Detailed Design

- يُعنى بكل مكون على حدة، من حيث:
  - **هياكل البيانات** المستخدمة داخله
  - **الخوارزميات** التي ينفذها
  - **واجهات البرمجة** APIs التي يقدّمها أو يستخدمها

# DESIGN VS. ARCHITECTURE



The design of any given SW encompasses two kinds of technical decisions; components design decisions and architectural decisions. Both of those decisions are key pillar in finishing the SW final blueprint/design.

Architecture decisions involve choices that impact the entire system, such as the selection of architectural patterns, the distribution of responsibilities, and the identification of key architectural styles.

Architecture Decisions are mostly related to fulfilling Non-Functional Requirement. This includes but not limited to; scalability, reliability, availability, maintainability.

يتضمن تصميم أي نظام برمجي نوعين أساسيين من **القرارات التقنية**، وكلاهما يُعد ركيزة أساسية في إتمام **المخطط النهائي للنظام**:

**قرارات التصميم المعماري** Architectural Decisions

تشمل القرارات التي تؤثر على **النظام ككل**، وتُحدد الإطار العام الذي يُبنى عليه المشروع. ومن أبرز هذه القرارات:

- اختيار **النمط المعماري** المناسب مثل Microservices أو Monolithic
- **توزيع المسؤوليات** بين المكونات المختلفة
- تحديد **الأساليب المعمارية الأساسية** التي توجه التطوير

هذه القرارات ترتبط غالبًا بـ **المتطلبات غير الوظيفية** Non-Functional Requirements، مثل:

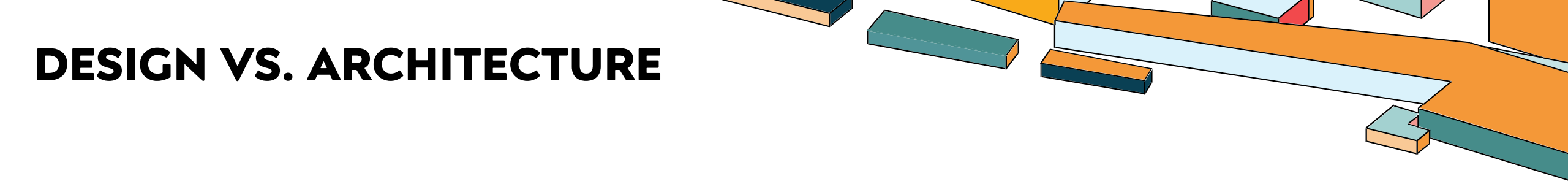
- القابلية للتوسع Scalability
- الاعتمادية Reliability
- التوفر Availability
- القابلية للصيانة Maintainability

**قرارات تصميم المكونات** Component Design Decisions

تركز على التفاصيل الدقيقة لكل مكون داخل النظام، وتشمل:

- اختيار **هياكل البيانات** المناسبة
- تحديد **الخوارزميات** التي ينفذها كل مكون
- تصميم **واجهات التفاعل** APIs بين المكونات

# DESIGN VS. ARCHITECTURE



Design decisions: are more granular and focus on the internal structure of components, algorithms, data structures, and interactions between classes or modules.

Design Decisions are mostly concerned with fulfilling the Functional requirements. This includes but not limited to; identifying the applications domain object model, business objects, and data entities that will be physically implemented through code writing.

NOTE: The architect unusually has more experience than designer.

**قرارات التصميم** تُعد أكثر تفصيلاً، إذ تركز على البنية الداخلية للمكوّنات، والخوارزميات، وهياكل البيانات، وكذلك على التفاعلات بين الأصناف أو الوحدات البرمجية. وتنصبّ هذه القرارات بشكل رئيسي على تلبية المتطلبات الوظيفية للنظام، ويشمل ذلك - دون أن يقتصر عليه - تحديد نموذج كائنات المجال الخاص بالتطبيق، والكائنات التجارية، والكيانات البيانية التي سيتم تنفيذها فعلياً من خلال كتابة الشيفرة البرمجية.

📌 **ملاحظة تنظيمية حول توزيع الخبرات داخل الفريق**

لوحظ أن **المهندس المعماري للنظام يمتلك خبرة عملية تفوق خبرة المصمم**، وهو ترتيب غير معتاد في بعض السياقات التطويرية التي يكون فيها المصمم أكثر دراية بتجربة المستخدم أو الجوانب الإبداعية.

**الانعكاسات المحتملة لهذا التوزيع:**

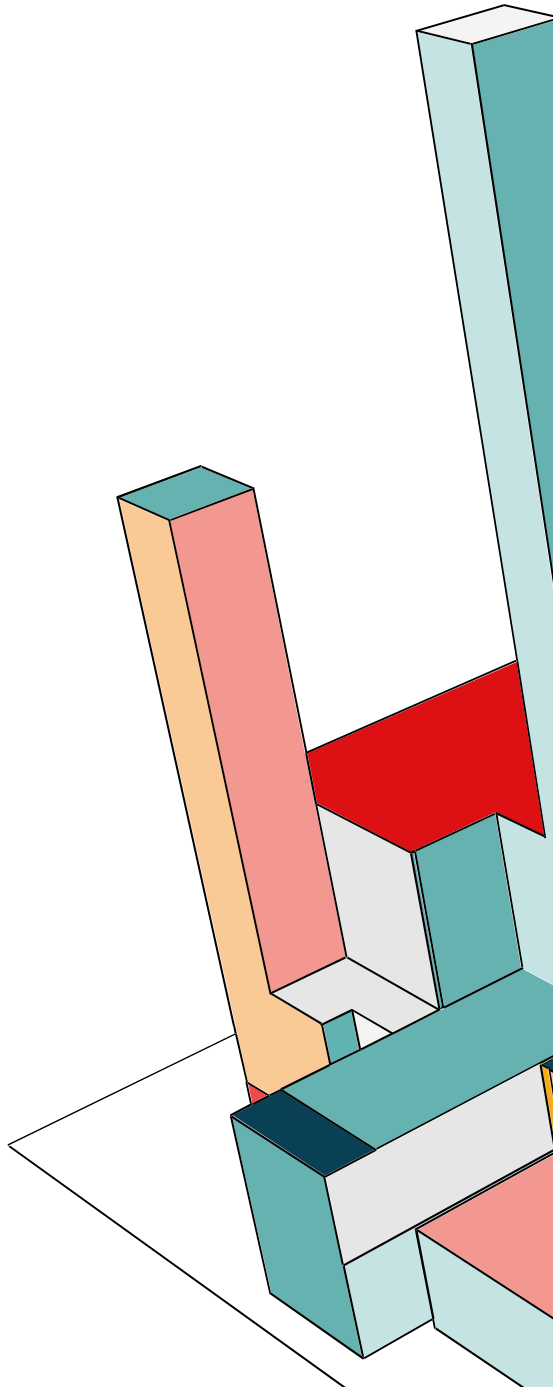
- قد يتولى المهندس المعماري دورًا أكثر فاعلية في توجيه قرارات التصميم لضمان توافقها مع البنية التقنية ومتطلبات النظام.
- يُتوقع أن يستفيد المصمم من التوجيه الفني والملاحظات البنيوية، خاصة في النقاط التي تتقاطع فيها تجربة المستخدم مع الأداء البرمجي.
- ينبغي أن تعتمد آلية التعاون بين الطرفين على تبادل المعرفة والتحقق المتبادل لضمان التوازن بين الابتكار التصميمي والاستقرار المعماري للنظام.

| Architecture | Design |
| --- | --- |
| Performed by architect. | Performed by designer. |
| high-level structures and the overall organization of the system. It focuses on major components or modules, their relationships, and how they interact at a high level. | delves into the details of individual components or modules. . making specific decisions about the internal structure of each component. |
| Architecture provides a blueprint for the entire system. It is more about the "big picture" and abstract concepts. | This deals with fine-grained details, addressing specific algorithms, data structures, and implementation details |
| <u>Some example decisions:</u><br>-Platform, operating system to be used.<br>-User experience technology (e.g. HTML5, CCS3, Silverlight, AJAX, etc...)<br>-Component messaging (e.g. SOAP). | <u>Some example decisions:</u><br>-Entity Relationship Diagram design.<br>-Class diagram design (in case of object Oriented programming)<br>-Classes data and behavior. |
| <u>Two main types of architect roles</u><br>- Application architect.<br>- Infrastructure architect. | <u>Three main types of designer roles</u><br>-User Interface (UI) designer.<br>-Application component designer.<br>-Data designer. |

# DESIGN VS. ARCHITECTURE

# محاور المحاضرة

- ما بين التصميم والهيكلة. Design vs. Architecture
- **أنواع أدوار المصمم والمعماري في هندسة البرمجيات** Design and Architecture Roles
- أنواع أدوار المصمم والمعماري في هندسة البرمجيات
- أهمية تحليل وجمع المتطلبات واثرها على عملية التصميم والهيكلة



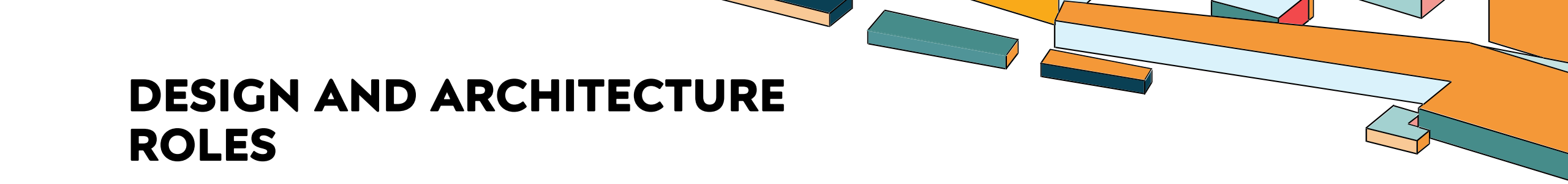
# ARCHITECTURAL DECISIONS

- How will the system be distributed?
- What architectural patterns or styles might be used?
- Is there a generic application architecture that can act as a template for the system that is being designed?
- What control strategy will be used to the operation of the components in the system?
- What architectural organization is best for delivering the non-functional requirements of the system?
- How will the architectural design be evaluated?
- How should the architecture of the system be documented?



**أسئلة معمارية لتصميم النظام البرمجي**

1. **كيف سيتم توزيع النظام؟** هل سيكون النظام مركزيًا، أم موزعًا عبر عدة خوادم أو خدمات؟ وهل سيتم نشره محليًا أم عبر السحابة؟
2. **ما الأنماط أو الأساليب المعمارية التي يمكن استخدامها؟** هل سيتم اعتماد نمط معماري مثل الطبقي Layered ، الخدمات المصغرة Microservices ، أو المعمارية الموجهة بالخدمات SOA ؟
3. **هل توجد بنية تطبيق عامة يمكن استخدامها كنموذج أولي؟** هل يمكن الاستفادة من نماذج معمارية جاهزة مثل ثلاثية الطبقات أو نموذج C4 لتسريع التصميم؟
4. **ما الاستراتيجية التي ستُستخدم للتحكم في تشغيل مكونات النظام؟** هل سيكون التحكم مركزيًا، أم موزعًا؟ وهل سيتم استخدام أحداث، حالات، أو مراقبين للتحكم في التفاعل بين المكونات؟
5. **ما التنظيم المعماري الأنسب لتحقيق المتطلبات غير الوظيفية؟** كيف يمكن للمعمارية المختارة أن تدعم الأداء، الأمان، القابلية للتوسع، والاعتمادية؟
6. **كيف سيتم تقييم التصميم المعماري؟** هل سيتم استخدام طرق مثل تحليل السيناريوهات، النمذجة الأولية، أو منهجية تحليل المفاضلات المعمارية ATAM ؟
7. **كيف يجب توثيق معمارية النظام؟** ما الأدوات والنماذج التي ستُستخدم لتوثيق البنية؟ هل سيتم استخدام مخططات مثل C4، أو قوالب مثل arc42؟ وما مستوى التفاصيل المطلوب؟



# DESIGN AND ARCHITECTURE ROLES

Types of Architect Roles:

- Application architect.
- Infrastructure/technical architect.

# APPLICATION ARCHITECT:

- He is the one who is concerned with defining the way that will be used for coupling, and cohering the application' s components, and layers.
- Application architect has to take major decisions that affect the rest of the development team, and the SW production cycles.
- Some decisions that are taken by AA, aligned with the gathered requirements and agreed scope and affect the development team:
  - ❑ OS, DBMS, UI types, Component Messaging,…
  - ❑ Platform to be utilized (on premise, cloud, hybrid),
  - ❑ The type of client devices (desktop PCs, tablets, mobile, etc…)
- ➢ One of the most important decisions is the Application Architectural Style or Architectural Pattern.

يُعد مهندس معمارية التطبيقات المسؤول الأول عن **تحديد الأسلوب الذي سيتم اعتماده في الربط** Coupling **والتماسك** Cohesion بين مكونات النظام وطبقاته المختلفة، بما يضمن تكاملًا وظيفيًا وهيكليًا فعالًا.

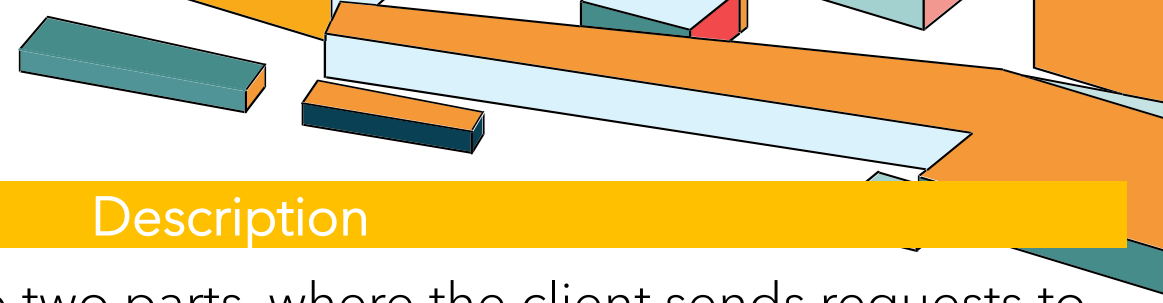
يتخذ مهندس المعمارية قرارات جوهرية تؤثر بشكل مباشر على باقي أعضاء فريق التطوير، وعلى **دورة إنتاج البرمجيات** بأكملها. وتُبنى هذه القرارات على المتطلبات التي تم جمعها والنطاق المتفق عليه للمشروع.

من أبرز القرارات التي يتخذها مهندس المعمارية، والتي تؤثر على الفريق التقني:

- ❑ اختيار **نظام التشغيل** Operating System المناسب.
- ❑ تحديد **نظام إدارة قواعد البيانات** DBMS .
- ❑ اختيار نوع واجهة المستخدم UI المناسبة للفئة المستهدفة.
- ❑ تحديد آلية **التراسل بين المكونات** Component Messaging .
- ❑ تحديد **المنصة التقنية** التي سيتم استخدامها: محلية On-Premise ، سحابية Cloud، أو هجينة Hybrid .
- ❑ تحديد نوع **الأجهزة العميلة** Client Devices التي سيدعمها النظام: الحواسيب المكتبية، الأجهزة اللوحية، الهواتف الذكية، وغيرها.

ومن أهم القرارات التي تقع ضمن مسؤوليات مهندس المعمارية هي **اختيار النمط المعماري** Architectural Style أو **النمط التصميمي المعماري** Architectural Pattern الذي سيُبنى عليه النظام، لما له من تأثير مباشر على قابلية التوسع، الأداء، وإدارة التعقيد البرمجي.

# APPLICATION ARCHITECT STYLES



| Architecture Style | Description |
|---|---|
| Client/ Server | Divide the application into two parts, where the client sends requests to the server. Usually, the server is a database with application logic represented as stored procedures. |
| Component-Based Architecture | Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces. |
| Layered Architecture | Separates the concerns of the application into stacked sets (layers). |
| N-Tiers/3Tiers | Divide functionality into loosely couple segments in a way that is close to the layered style, however in this style, each segment is being a tier that is usually deployed on a physically separate machine. |
| Model View Controller (MVC) | MVC separates an application into three interconnected components, each with its own distinct role and responsibility |

# INFRASTRUCTURE/ TECHNICAL ARCHITECT:

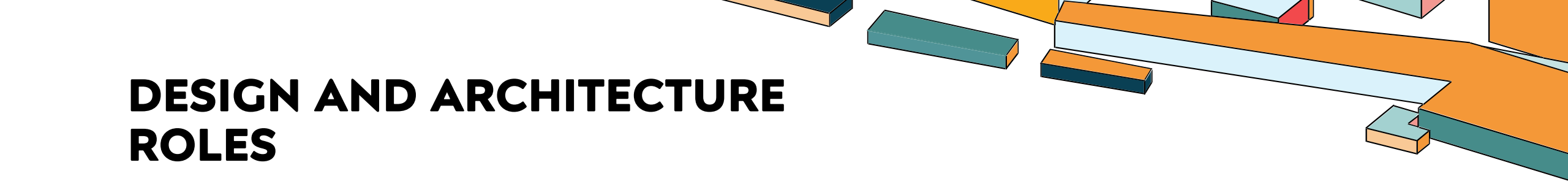
- Infrastructure architect is the one who defines how these components and layers should be physically deployed on the hardware, and network infrastructure.
- Infrastructure architect decisions may include; defining the security zones, defining the kinds of server clusters that will be used.
- Some Quality Attributes are used.

يتولى مهندس البنية التحتية مسؤولية **تحديد كيفية نشر مكونات النظام وطبقاته فعليًا على مستوى العتاد** Hardware **والبنية الشبكية** Network Infrastructure ، بما يضمن تكاملًا آمنًا وفعالًا بين العناصر التقنية المختلفة.

تشمل قرارات مهندس البنية التحتية ما يلي:

- **تحديد المناطق الأمنية** Security Zones داخل النظام، لضمان العزل المناسب بين المكونات الحساسة والمكونات العامة.
- **اختيار أنواع مجموعات الخوادم** Server Clusters التي سيتم استخدامها، بما يتناسب مع متطلبات الأداء والتوافر.
- **تطبيق سمات الجودة** Quality Attributes مثل القابلية للتوسع، التوافر العالي، الأمان، وقابلية الصيانة، لضمان استقرار النظام على المدى الطويل.

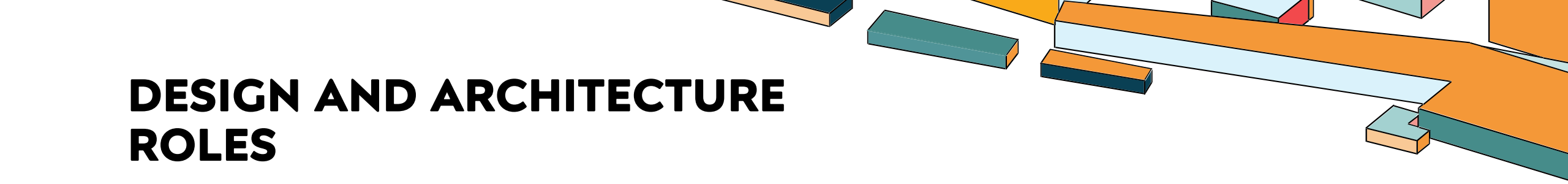
تُعد هذه القرارات أساسية في ضمان أن البنية التحتية للنظام تدعم الأهداف المعمارية والتشغيلية للتطبيق، وتتكامل بسلاسة مع القرارات التي يتخذها مهندس معمارية التطبيقات



# DESIGN AND ARCHITECTURE ROLES

Types of Designer Roles :

- User Interface (UI) designer
- Application domain designer
- Data designer



# DESIGN AND ARCHITECTURE ROLES

## Types of Designer Roles :

- **User Interface (UI) designer**

  UI design is out of the scope. You can refer to Human- Computer-Interaction (HCI) Course for more extra details related this topic.

يُعد تصميم واجهة المستخدم UI من المواضيع المهمة في تطوير الأنظمة، إلا أنه **يخرج عن نطاق هذه المادة الدراسية**. لذلك، لن يتم التطرق إلى تفاصيل تصميم الواجهات ضمن محتوى هذا المقرر.

لمن يرغب في التوسع في هذا المجال، يمكن الرجوع إلى مقرر **التفاعل بين الإنسان والحاسوب** Human-Computer Interaction – HCI ، حيث يتم تناول مفاهيم تصميم الواجهات، تجربة المستخدم (UX)، وعوامل التفاعل البصري والسلوكي بشكل أكثر تفصيلًا

# TYPES OF DESIGNER ROLES :

- ## Application domain designer

- ❑ Domain designer is a major key player in the process of architecting and designing an application.
- ❑ He is responsible of understanding the detailed business rules, and the main domain objects that are generated from this set of customer' s business rule requirements.
- ❑ He designs the application components that have enough algorithms to maintain these requirements processed and implemented accurately.

يُعد مصمم نطاق التطبيق أحد العناصر الأساسية في عملية **تصميم وهيكلة النظام البرمجي**، حيث يلعب دورًا محوريًا في الربط بين المتطلبات التجارية للمستخدمين والبنية التقنية للتطبيق.

**المسؤوليات الرئيسية:**

- ❑ فهم **القواعد التجارية التفصيلية** الخاصة بالمجال الذي يستهدفه التطبيق.
- ❑ تحديد وصياغة **الكائنات النطاقية الأساسية** Domain Objects الناتجة عن تحليل متطلبات الأعمال.
- ❑ تصميم **مكونات التطبيق** التي تحتوي على خوارزميات كافية لمعالجة هذه المتطلبات وتنفيذها بدقة وفعالية.

يقوم مصمم النطاق بتحويل المتطلبات التجارية إلى نماذج قابلة للتنفيذ برمجيًا، مما يضمن أن النظام يعكس الواقع العملي للمستخدمين ويستجيب لاحتياجاتهم بشكل مباشر.

# TYPES OF DESIGNER ROLES :

- **Data designer**

Those are a kind of database (DB) professionals who are concerned with designing the applications DB, and defining its chosen schema; logically and physically.

يُعد مصمم البيانات أحد المتخصصين في قواعد البيانات، ويُعنى بشكل أساسي بـ **تصميم قاعدة بيانات التطبيق**، من خلال تحديد **المخطط البنيوي** Schema المناسب لها على المستويين **المنطقي** Logical و**الفيزيائي** Physical

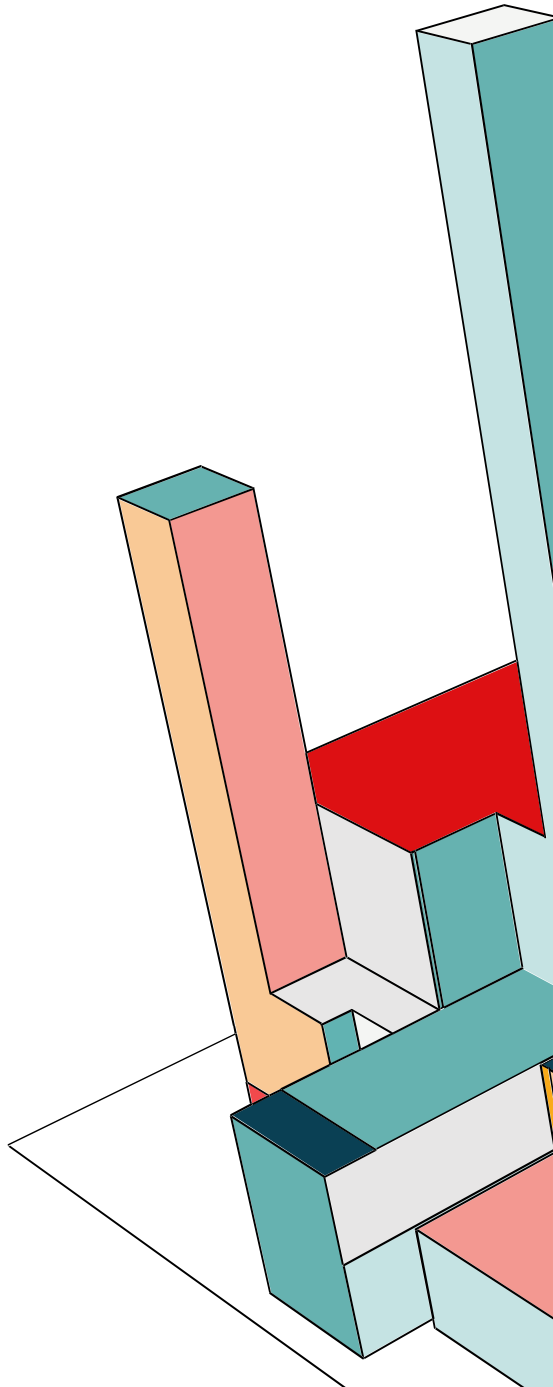
**المهام الأساسية:**

- ❑ تحليل متطلبات البيانات المستخلصة من نطاق التطبيق.
- ❑ تصميم الكيانات والعلاقات والحقول بما يتوافق مع احتياجات النظام.
- ❑ تحديد البنية المنطقية للبيانات (مثل الجداول، المفاتيح، العلاقات).
- ❑ تصميم البنية الفيزيائية لتخزين البيانات بما يحقق الأداء والكفاءة.
- ❑ التعاون مع مهندس المعمارية لضمان تكامل قاعدة البيانات مع باقي مكونات النظام.

يلعب مصمم البيانات دورًا محوريًا في ضمان أن تكون بنية البيانات **متماسكة، قابلة للتوسع، وآمنة**، مما يسهم في استقرار النظام وفعاليته على المدى الطويل.

# محاور المحاضرة

- ما بين التصميم والهيكلة. Design vs. Architecture
- **أنواع أدوار المصمم والمعماري في هندسة البرمجيات** Design and Architecture Roles
- أنواع أدوار المصمم والمعماري في هندسة البرمجيات
- Architectural views



# ARCHITECTURAL VIEWS

- It is impossible to represent all relevant information about a system's architecture in a single architectural model, as each model only shows one view or perspective of the system.
- It might show how a system is decomposed into modules, how the run-time processes interact, or the different ways in which system components are distributed across a network.
- All of these are useful at different times so, for both design and documentation, you usually need to present multiple views of the software architecture.

من غير الممكن تمثيل جميع المعلومات المتعلقة بمعمارية النظام في نموذج معماري واحد، وذلك لأن كل نموذج يُظهر **وجهة نظر أو منظورًا معينًا** فقط من جوانب النظام.

فقد يُظهر النموذج كيفية **تفكيك النظام إلى وحدات** Modules ، أو كيفية **تفاعل العمليات أثناء وقت التشغيل** Run-Time Processes ، أو الطرق المختلفة التي يتم بها **توزيع مكونات النظام عبر الشبكة**.

تُعد هذه النماذج المختلفة مفيدة في مراحل متعددة من دورة حياة النظام، سواء في **مرحلة التصميم** أو أثناء **توثيق المعمارية**. لذلك، من الضروري تقديم **عدة وجهات نظر** Multiple Views للمعمارية البرمجية، لضمان فهم شامل ودقيق لجميع جوانب النظام

# ARCHITECTURAL VIEWS

There are different opinions as to what views are required. philippe Krutchen (1995), in his well-known 4+1 view model of software architecture, suggests that there should be four fundamental architectural views, which are related using use cases or scenarios.

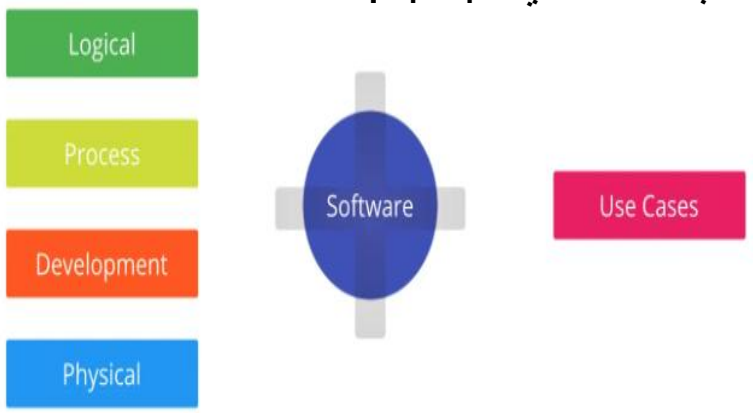
The reason behind the name: 4+1

The model has four views: logical, development, process, and physical. In addition, selected use cases or scenarios are utilized as the 'plus one' view to show the design. As a result, the model has 4+1 views. Hence the model is called The 4+1 Architectural View.

**نموذج 4+1 للمعمارية البرمجية** – لـ Philippe Kruchten (1995)

توجد آراء متعددة حول عدد وأنواع وجهات النظر Views التي يجب تضمينها عند تصميم معمارية النظام البرمجي. وقد اقترح Philippe Kruchtenفي عام 1995 نموذجًا شهيرًا يُعرف باسم **نموذج 4+1 للمعمارية البرمجية**، والذي يُعد من أكثر النماذج استخدامًا في توثيق وتصميم الأنظمة المعقدة.

**سبب التسمية "4+1":** يتكون النموذج من **أربعة وجهات نظر أساسية**، بالإضافة إلى وجهة نظر خامسة تُستخدم لربط هذه الوجهات من خلال **حالات الاستخدام** Use Cases أو **السيناريوهات** Scenarios ، ومن هنا جاءت التسمية "4+1".




يساعد نموذج 4+1 في تقديم **رؤية شاملة ومتعددة الأبعاد** لمعمارية النظام، مما يسهل فهمه من قبل مختلف أصحاب المصلحة (المطورين، المعماريين، المستخدمين، ومديري المشاريع)، ويعزز جودة التصميم والتوثيق.

# 4 + 1 ARCHITECTURAL VIEW MODEL

1. A logical view:

- which shows the key abstractions in the system as objects or object classes. It should be possible to relate the system requirements to entities in this logical view.
- Focuses on the functionality of the system.
- Represents the software components and their interactions without detailing the implementation.
- Represents the hierarchy and structure of the software components.
- UML diagrams are used to represent the logical view, and include class diagrams, and state diagrams.
- The Logical View helps in understanding the system's structure and functionality without getting into the implementation details.
- Viewer: End User.

Note: Logical View: The functionality. The service.

يمثل المنظور المنطقي أحد المكونات الأساسية في نماذج المعمارية البرمجية، حيث يُستخدم لعرض **التجريدات الرئيسية في النظام** على شكل كائنات أو أصناف كائنية Object Classes ويُفترض أن يكون بالإمكان **ربط متطلبات النظام** بالكائنات الممثلة في هذا المنظور.

**الخصائص الرئيسية للمنظور المنطقي:**

- يركز على **الوظائف الأساسية للنظام** دون الدخول في تفاصيل التنفيذ.
- يُظهر **مكونات النظام البرمجية** وطريقة تفاعلها، مع الحفاظ على مستوى عالٍ من التجريد.
- يُبرز **الهيكلية الهرمية والتنظيمية** للمكونات البرمجية.
- يُستخدم في تمثيله **مخططات** UML مثل:
  - مخطط الأصناف Class Diagram
  - مخطط الحالات State Diagram

**الفائدة الأساسية:** يساعد المنظور المنطقي في **فهم بنية النظام ووظائفه** بطريقة منظمة، دون الحاجة إلى التعمق في تفاصيل التنفيذ البرمجي، مما يجعله مناسبًا لعرض النظام على أصحاب المصلحة غير التقنيين مثل المستخدمين النهائيين.

**المستفيد الأساسي من هذا المنظور: المستخدم النهائي** End User

يركز المنظور المنطقي على **الوظائف الأساسية للنظام** وعلى **الخدمات التي يقدمها** للمستخدمين النهائيين. فهو يُعنى بتحديد ما يقوم به النظام من مهام، دون التطرق إلى كيفية تنفيذها تقنيًا، مما يجعله حجر الأساس لفهم سلوك النظام من منظور وظيفي وتجريدي.

# 4 + 1 ARCHITECTURAL VIEW MODEL

2. A process view:

- which shows how, at run-time, the system is composed of interacting processes. This view is useful for making judgments about non- functional system characteristics such as performance and availability.
- It explains the system processes and how they communicate.
- Captures the concurrency aspects of the design
- distribution, performance, and scalability are all addressed in the process view.
- The sequence diagram, communication diagram, and activity diagram are all UML diagrams that can be used to describe a process view.

يعرض المنظور المعالجي كيفية **تكوين النظام أثناء وقت التشغيل** من مجموعة من العمليات المتفاعلة، ويُستخدم هذا المنظور لتقييم الخصائص غير الوظيفية للنظام مثل **الأداء** Performance و**التوافر** Availability

**الخصائص الرئيسية للمنظور المعالجي:**

- يوضح **العمليات التي ينفذها النظام** وكيفية **تواصلها وتفاعلها** فيما بينها.
- يُبرز الجوانب المتعلقة بـ **التوازي** Concurrency في التصميم.
- يتناول قضايا **التوزيع** Distribution ، و**قابلية التوسع** Scalability ، و**كفاءة الأداء**.
- يُستخدم في تمثيله مجموعة من **مخططات** UML، منها:
  - مخطط التسلسل Sequence Diagram
  - مخطط التواصل Communication Diagram
  - مخطط النشاط Activity Diagram

**ملاحظة: المنظور المعالجي يُعنى بالتواصل بين العمليات والخدمات داخل النظام.**



Note: Process View: Communication between processes and/or services.

# 4 + 1 ARCHITECTURAL VIEW MODEL

## 3. A development view:

- It shows how the software is decomposed for development, that is, it shows the breakdown of the software into components that are implemented by a single developer or development team.
- Describes the organization of the software in its development environment. This view can be modelled with UML´s component and package diagrams.
- This view is useful for software managers and programmers.

يعرض المنظور التطويري كيفية **تفكيك النظام البرمجي إلى مكونات قابلة للتنفيذ والتطوير**، بحيث يتم توزيع هذه المكونات على فرق التطوير أو المطورين بشكل فردي. يركز هذا المنظور على **تنظيم النظام داخل بيئة التطوير**، ويُستخدم لتوضيح البنية الداخلية للمشروع من منظور هندسي وتقني.

**الخصائص الرئيسية للمنظور التطويري:**

- يُظهر تقسيم النظام إلى **وحدات برمجية مستقلة** يمكن تطويرها بشكل منفصل.
- يصف **تنظيم الملفات والمكونات** داخل بيئة التطوير (مثل الحزم، المكتبات، الوحدات).
- يُستخدم في تمثيله **مخططات** UMLمثل:
  - مخطط المكونات Component Diagram
  - مخطط الحزم Package Diagram

**الفئة المستفيدة من هذا المنظور: مديرو المشاريع البرمجية والمبرمجون**، حيث يساعدهم في فهم توزيع المهام، إدارة الكود، وتنسيق العمل بين الفرق.

# 4 + 1 ARCHITECTURAL VIEW MODEL

## 4. A physical view:

- It shows the system hardware and how software components are distributed across the processors in the system. This view is useful for systems engineers planning a system deployment.
- Describes the mapping of the software onto the hardware. Deployment diagrams are used to model this view.

يعرض المنظور الفيزيائي **البنية المادية للنظام**، حيث يُوضح مكونات العتاد Hardware وكيفية **توزيع المكونات البرمجية** على المعالجات أو الخوادم داخل النظام. يُستخدم هذا المنظور بشكل أساسي من قبل **مهندسي الأنظمة** عند التخطيط لنشر النظام وتشغيله في بيئة الإنتاج.

**الخصائص الرئيسية للمنظور الفيزيائي:**

- يصف **كيفية ربط البرمجيات بالعتاد**، أي كيف يتم نشر مكونات النظام فعليًا على الأجهزة.
- يُستخدم لتحديد مواقع التنفيذ، توزيع الحمل، والتخطيط للبنية التحتية.
- يُعالج قضايا مثل **التوافر، الأداء، والتوزيع الجغرافي** للمكونات.
- يُستخدم في تمثيله **مخطط النشر** Deployment Diagram ضمن نماذج UML

**الفئة المستفيدة من هذا المنظور: مهندسو الأنظمة** Systems Engineers ، حيث يساعدهم في اتخاذ قرارات تتعلق بتوزيع الموارد، إعداد الخوادم، وضمان جاهزية البيئة التشغيلية.

# 4 + 1 ARCHITECTURAL VIEW MODEL

- The +1 comes in from the scenarios view which is what your end users actually care about. It' s the system functionality/capabilities
- Use Case View: It is illustrated by selected use cases or scenarios. It contains diagrams describing what the system does from a black box perspective. This view contains use case diagrams.

**المنظور الخامس (+1): منظور حالات الاستخدام** ((Use Case View

يمثل منظور حالات الاستخدام العنصر المكمل لنموذج 4+1، ويُشار إليه بـ "+1" لأنه يُستخدم لربط وتوضيح العلاقة بين المنظورات الأربعة الأخرى من خلال **السيناريوهات الواقعية** التي تهم المستخدمين النهائيين بشكل مباشر.

**الخصائص الرئيسية لهذا المنظور:**

- يُركز على **وظائف النظام وقدراته** من وجهة نظر المستخدم.
- يُمثل النظام كـ **صندوق أسود**، حيث يتم وصف ما يقوم به دون التطرق إلى كيفية التنفيذ الداخلي.
- يُستخدم في تمثيله **مخططات حالات الاستخدام** use Case Diagrams ضمن نماذج UML.
- يُوضح **السيناريوهات التفاعلية** التي توضح كيف يتفاعل المستخدم مع النظام لتحقيق أهداف معينة.

**الفئة المستفيدة من هذا المنظور: المستخدمون النهائيون وأصحاب المصلحة غير التقنيين**، حيث يساعدهم في فهم ما يقدمه النظام من خدمات ووظائف بشكل مبسط وواضح.

# 4 + 1 VIEW MODEL OF SOFTWARE ARCHITECTURE

## Benefits of 4+1:

Better organization with better separation of concern

The 4+1 maps stakeholders to the information they need, without requiring specific notations.

**فوائد نموذج 4+1**

•**تنظيم أفضل وفصل واضح للاهتمامات (**Separation of Concern): يتيح النموذج تقسيم المعمارية إلى وجهات نظر مستقلة، مما يسهل فهم كل جانب من جوانب النظام على حدة.

•**مواءمة أصحاب المصلحة مع المعلومات المناسبة:** يوفر لكل فئة من أصحاب المصلحة (مثل المطورين، المستخدمين، المهندسين) العرض المناسب الذي يلبي احتياجاتهم، دون الحاجة إلى استخدام رموز أو تقنيات تمثيل خاصة.

## Drawbacks:

4+1 was created in 1995, where software development was very different than today, there was no need for more than 2 or 3 views.

Nowadays, there are many complex systems which can´t be represented correctly with 4+1.

**محدوديات نموذج 4+1**

•**النموذج قديم نسبيًا:** تم تطوير نموذج 4+1 في عام 1995، في فترة كانت فيها بيئات تطوير البرمجيات أقل تعقيدًا، ولم تكن الحاجة ملحة لأكثر من وجهتي نظر أو ثلاث.

•**عدم كفاية النموذج لتمثيل الأنظمة الحديثة المعقدة:** في الوقت الحالي، توجد أنظمة برمجية ذات بنى معمارية متقدمة ومتعددة الطبقات، يصعب تمثيلها بدقة باستخدام نموذج 4+1 فقط، مما يستدعي استخدام نماذج أو وجهات نظر إضافية لتغطية الجوانب الأمنية، التشغيلية، والتكاملية.

**توجد عدت نماذج أخرى بديلة أو مكملة لنموذج 4+1 تشمل نماذج متعددة تهدف إلى معالجة تعقيدات الأنظمة الحديثة، مثل نموذج C4، ونموذج Views and Beyond، ونموذج Zachman، وغيرها. كل منها يقدم وجهات نظر إضافية أو مختلفة لتغطية الجوانب التي لا يشملها نموذج 4+1.**

أبرز النماذج المعمارية البديلة أو المكملة:

**نموذج** Zachman Framework

•يُستخدم في هندسة المؤسسات (Enterprise Architecture).

•يتكون من **مصفوفة من 6 أسئلة (ماذا، كيف، أين، من، متى، ولماذا)** مقابل 6 مستويات من الاهتمام (من المخطط إلى التشغيل).

•يُركز على **الربط بين الأعمال والتقنية**، ويُستخدم لتوثيق الأنظمة من زوايا متعددة.

**نموذج** C4 (Context, Containers, Components, Code).

•يُستخدم لتوثيق معمارية البرمجيات بطريقة مرئية ومنظمة.

•يتكون من أربع مستويات:

Context: يوضح كيف يتفاعل النظام مع البيئة الخارجية.

Containers: يوضح الحاويات البرمجية مثل التطبيقات وقواعد البيانات.

Components: يوضح مكونات كل حاوية.

Code: يوضح تفاصيل الكود البرمجي (اختياري).

•يُعد مكملًا فعالًا لنموذج 4+1 لأنه يركز على التفاصيل التقنية والتنفيذية.

**نموذج** Views and Beyond.

•طُوّر بواسطة Paul Clements وآخرين في كتاب Software Architecture in Practice.

•لا يحدد عددًا ثابتًا من المنظورات، بل يعتمد على **احتياجات أصحاب المصلحة**.

•يشجع على استخدام أي عدد من المنظورات مثل:

- منظور الأمان
- منظور الصيانة
- منظور الأداء

•يُعد أكثر مرونة من نموذج 4+1، ويُستخدم في الأنظمة المعقدة متعددة المتطلبات.

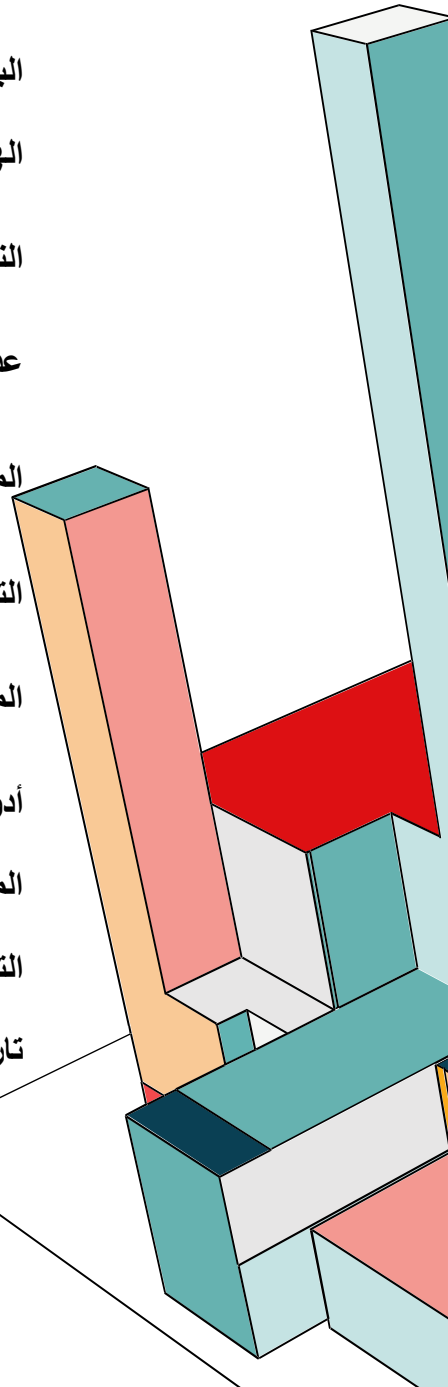
**نموذج** TOGAF (The Open Group Architecture Framework)

•إطار عمل شامل لهندسة المؤسسات.

•يُستخدم لتصميم وتخطيط وتنفيذ وإدارة بنية الأنظمة.

•يحتوي على منهجية ADM (Architecture Development Method) التي توجه مراحل تطوير المعمارية

| البند | نموذج TOGAF الإطار الخامس | نموذج 4+1 المعماري |
|---|---|---|
| الهدف الأساسي | تطوير وإدارة معمارية المؤسسات بشكل شامل | توثيق وتصميم معمارية البرمجيات من خلال وجهات نظر متعددة |
| النطاق | يشمل المؤسسة بالكامل: الأعمال، التطبيقات، البيانات، التقنية | يركز على النظام البرمجي فقط |
| عدد المنظورات | مرن ويشمل عدة مجالات عبر دورة تطوير المعمارية ADM | 5 (4 وجهات نظر + حالات الاستخدام) |
| المنهجية | منهجية قابلة للتخصيص حسب احتياجات المؤسسة ADM | ثابتة ومحددة مسبقًا |
| التركيز الفني | الأعمال، الحوكمة، القدرات، التكامل، إدارة التغيير | الوظائف، العمليات، التطوير، النشر، الاستخدام |
| المستفيدون الرئيسيون | مدراء الأعمال، مهندسو المؤسسات، فرق الحوكمة والتخطيط | المطورون، المعماريون، المستخدمون النهائيون |
| أدوات التمثيل | لا يعتمد على أدوات تمثيل محددة، يمكن استخدام ArchiMateأو UML | UML مخططات الأصناف، النشاط، النشر، التسلسل |
| المرونة والتوسع | عالي المرونة وقابل للتوسع في بيئات متعددة | محدود في الأنظمة المعقدة الحديثة |
| التكامل مع الأطر الأخرى | يمكن دمجه مع أطر مثل ITIL، COBIT، BPMN | مستقل نسبيًا |
| تاريخ الإنشاء | 1995 – بواسطة The Open Group | 1995 – بواسطة Philippe Kruchten |



- نموذج 4+1 مناسب لتوثيق وتصميم الأنظمة البرمجية من منظور هندسي وتقني، ويُستخدم غالبًا في المشاريع البرمجية متوسطة التعقيد.
- **TOGAF** يُعد إطارًا شاملاً لهندسة المؤسسات، ويُستخدم في التخطيط الاستراتيجي، إدارة التغيير، وتكامل الأنظمة على مستوى المؤسسة.

# APPENDIXES

# REFERENCES


| رقم المصدر | سنة النشر | المؤلف | عنوان المحتوى |
|---|---|---|---|
| 1 | 2025 | Hironori Washizaki | Guide to the Software Engineering Body of Knowledge v4.0 |
| 2 | 2023 | Marwa Solla | Software architecture and design for modern large-scale systems |
| 3 | 2024 | akkah.com بكه للتعليم | الفرق بين المتطلبات الوظيفية وغير الوظيفية وتقنيات استنباطهما وأفضل الممارسات |
| 4 | 2006 | Christopher John Fox | Introduction to Software Engineering Design Processes, Principles, and Patterns with UML2 |
| 5 | 2025 | Wikipedia | ISO/IEC 12207 - Wikipedia |

# ASSIGNMENT 2

إجراء بحث حول أفضل التطبيقات أو الممارسات التقنية الموجهة للأطفال، والتي تهدف إلى تعزيز التعلم، التفاعل الإيجابي، أو تقليل الإدمان الرقمي.
يجب أن يتضمن البحث تحليلًا نقديًا لتصميم هذه التطبيقات، ومدى توافقها مع مبادئ التصميم الأخلاقي والآمن للأطفال.

**المتطلبات:**

اختيار تطبيقين إلى ثلاثة تطبيقات كمثال.

تحليل عناصر التصميم (الواجهة، التفاعل، المحتوى، الألوان، المكافآت).

تقييم مدى ملاءمة التطبيقات للفئة العمرية المستهدفة.

تقديم توصيات تقنية وتصميمية مستخلصة من التحليل.

**عدد الصفحات:** من 4 إلى 6 صفحات.