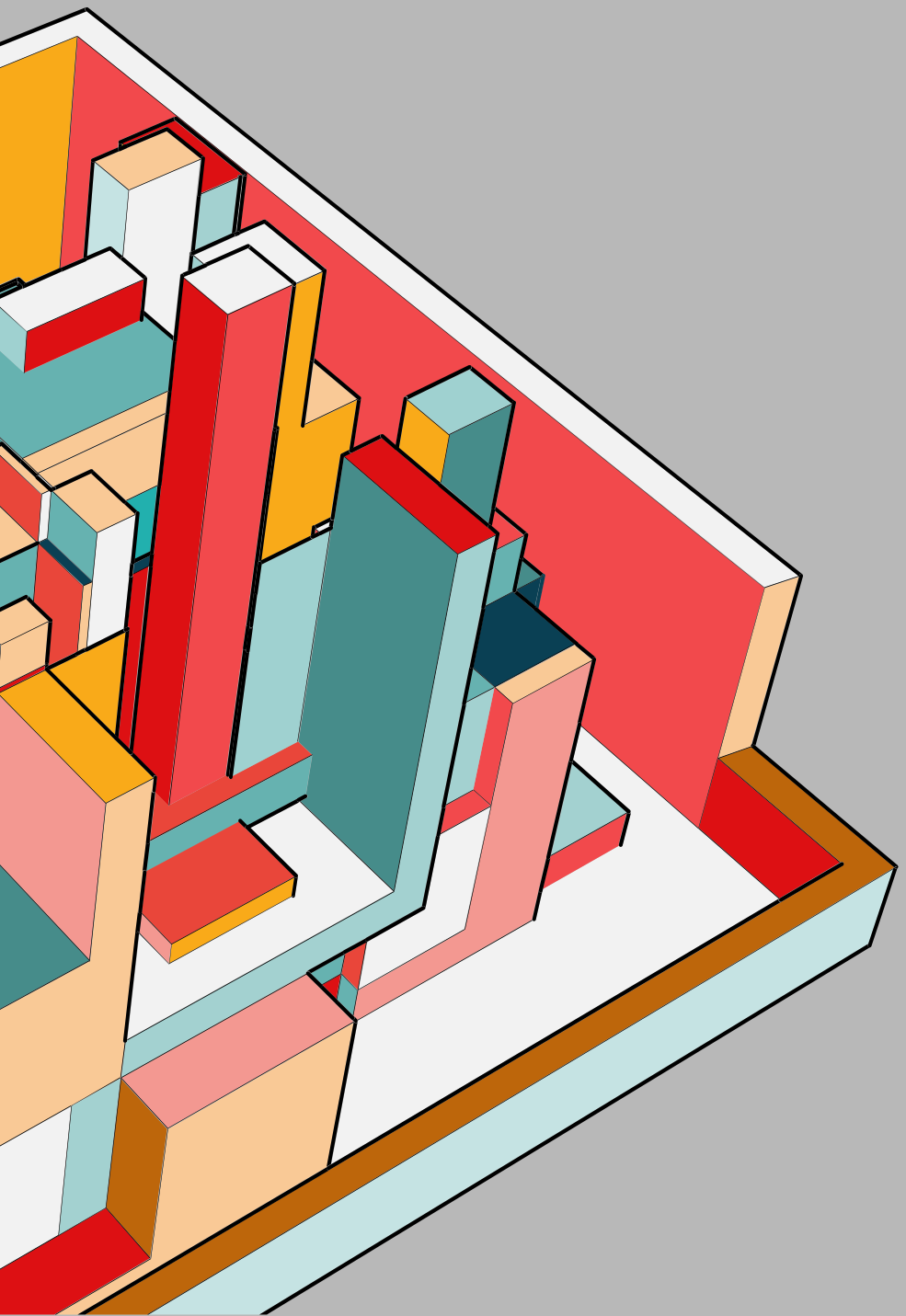


تصميم وهيكلة البرمجيات
ITSE411

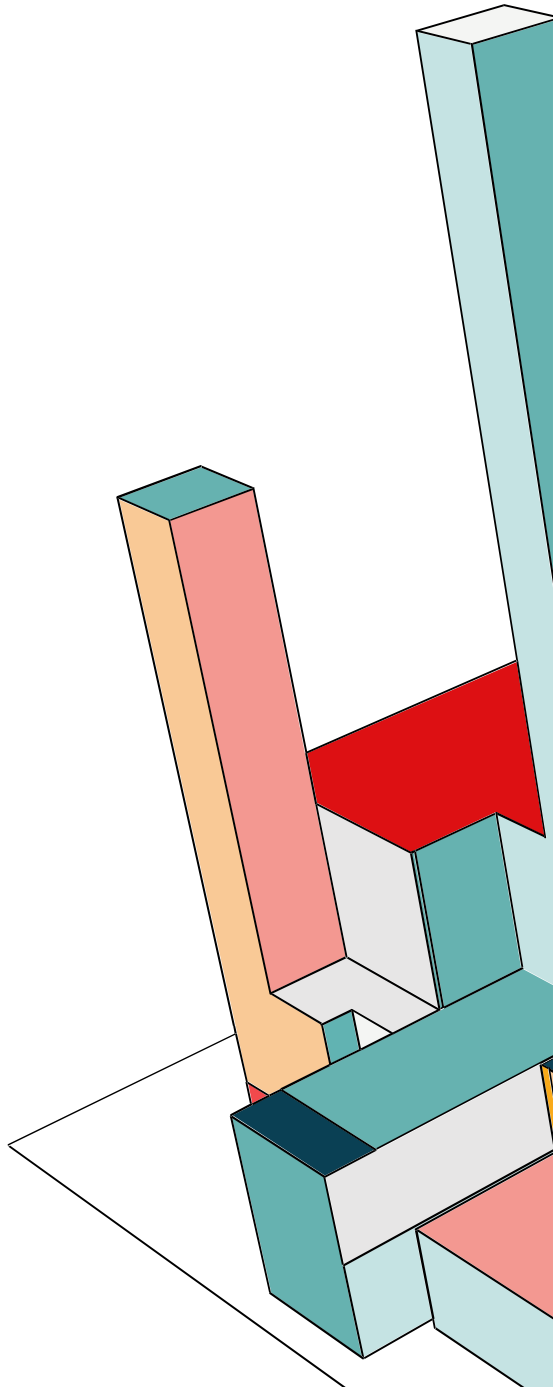
المحاضرة الأولى

تقـــــديم

بيرم زرتي - 2025

محاور المحاضرة

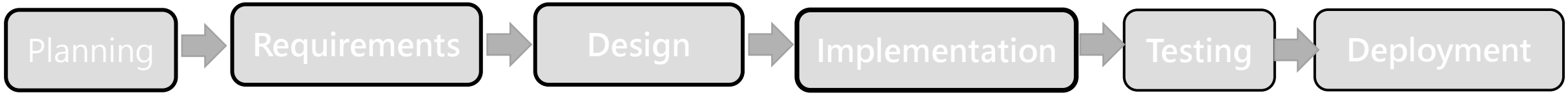
- **مقدمة ونظرة عامة عن مراحل تطوير النظم** Software development Cycle
- مرحلة التصميم أهميتها وترتيبها في دورة تطوير البرمجيات. Importance of Design
  - عمليات التصميم والهيكلة
- أهمية تحليل وجمع المتطلبات وعلاقتها بمرحلة التصميم والهيكلة
  - المتطلبات الوظيفية
  - المتطلبات الغير وظيفية
  - قيود النظام
- الملاحق
  - المراجع
  - واجب



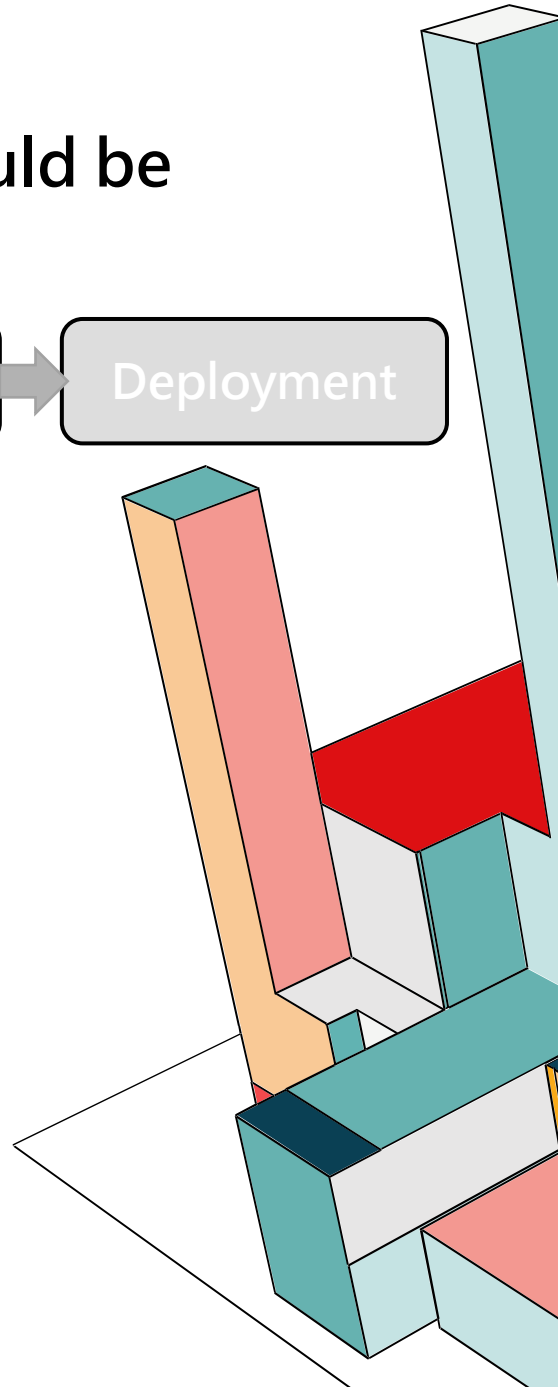
IS SOFTWARE DEVELOPMENT IS A REAL INDUSTRY?

- In order to understand if software (SW) development is an **industry, or not**, and how does it work, we need to understand the six main activities of any given industry. The main question is that is software Development is a real industry?

Each, and every industry worldwide has main activities that should be accomplished.



1. Planning.
2. Requirements Analysis.
3. Architecture and Design.
4. Build.
5. Test.
6. Deliver/Deploy.



EXAMPLE 1: LET US TAKE THE PLANES PRODUCTION INDUSTRY AS AN EXAMPLE

- If Boeing will produce a plane for Libyan Airlines, first of all, they have to understand how Libyan Airlines thinks in the potential plane. This needs to reach answers for lots of questions including:

  - What is the range of that plane?
  - How many seats should be in that plane?
  - Which engine Libyan needs the plane to come with?
  - What is the dedicated budget in Libyan Air for that plane?

EXAMPLE 1: PLANES PRODUCTION INDUSTRY

- The previous questions helps in **defining the scope** of that project. Answers helps in having a clear consensus about if this production project could be conducted or not.
- If there some technical, legal, expertise- related, political, or other could arise, and couldn’ t be mitigated, then the production project is not feasible.
- If the dedicated budget was not reasonable at all then, Boeing will notify Libyan Airline that this plane couldn’ t be produced and delivered to Libya.
- In contrary, if the project shows feasibility then, the rest of information will help in defining the scope of the project.

- Scope sets the boundaries of the project from every prospective that shouldn’ t be exceeded, including but, not limited to; timing, budgetary, resources.

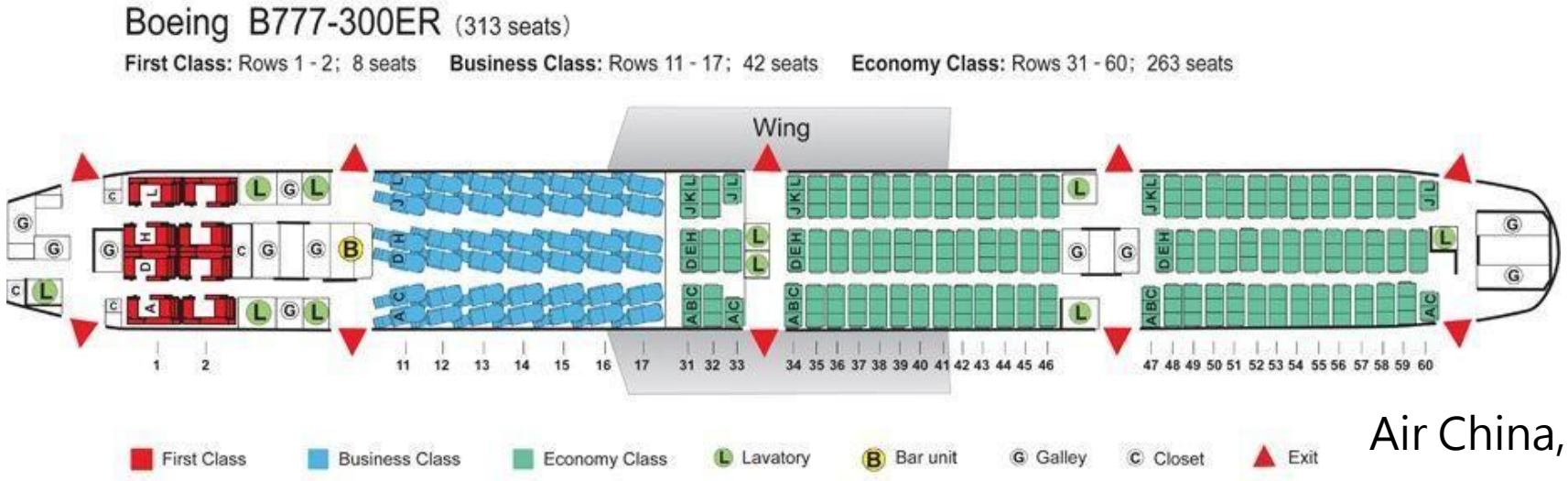
- The scope in our context could be the type of the plane (e,g 777, 737,747,..etc). If they have chosen a 777 plane. The subtype of the plane must to be defined as well (777-200, 777-300, 777-300ER, 777-400 or 777- 400ER)?

- Without such kind of information, the scope couldn’ t be defined clearly. Moreover, it may lead to complete failure of the project due to failure in achieving all what has been agreed on during scope definition.

- After contract agreement between Boeing and Libyan Airline, the second activity should begin. During this activity, Boeing should **gather the requirements, or wish list** of Libyan Airline in this plane.
- This wish list include answers for:
  - What will be the material of furnishing the seats (e.g. recycled leather, genuine leather, textiles, etc…)?
  - How many first-class, business-class, and economy class seats should be placed?
  - What will be spacing distance between each class' s seats?
  - How will each seat be equipped (e.g. LCD screen, Sky Phone, First Class bed-like seat, etc…)?
- This activity is called **Requirements Analysis.**

- When both parties finally agree on all requirements, Boeing should pass the agreed specifications to the manufacturing team.
- This team is concerning with producing the product (the Plane in our example). In order to produce the plane, a blueprint should be there for the people who will build it.
  - The blueprint has lots of details that should be elaborated in order to let the producers able to build the plane.
- Usually, the blueprint includes two important parts;

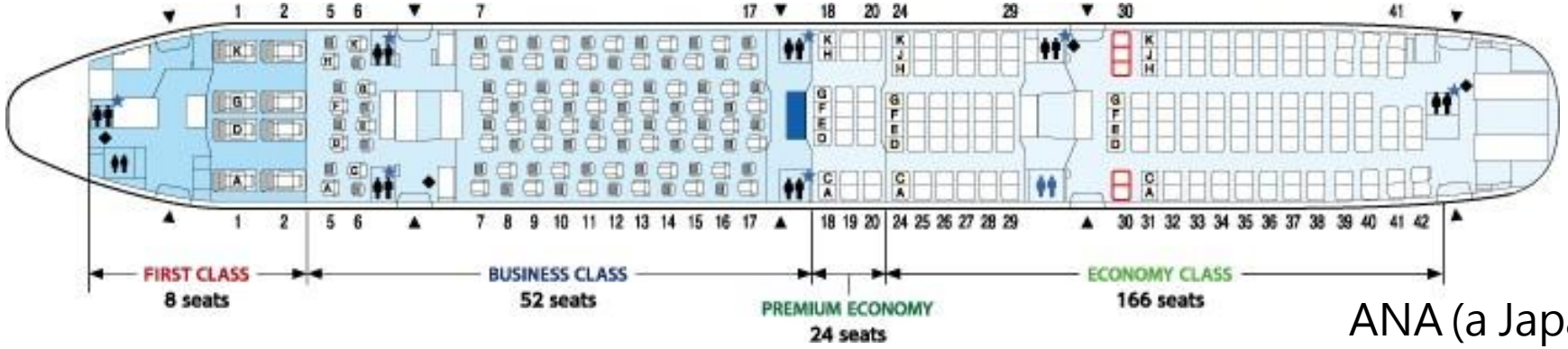
the product Architecture & Design.



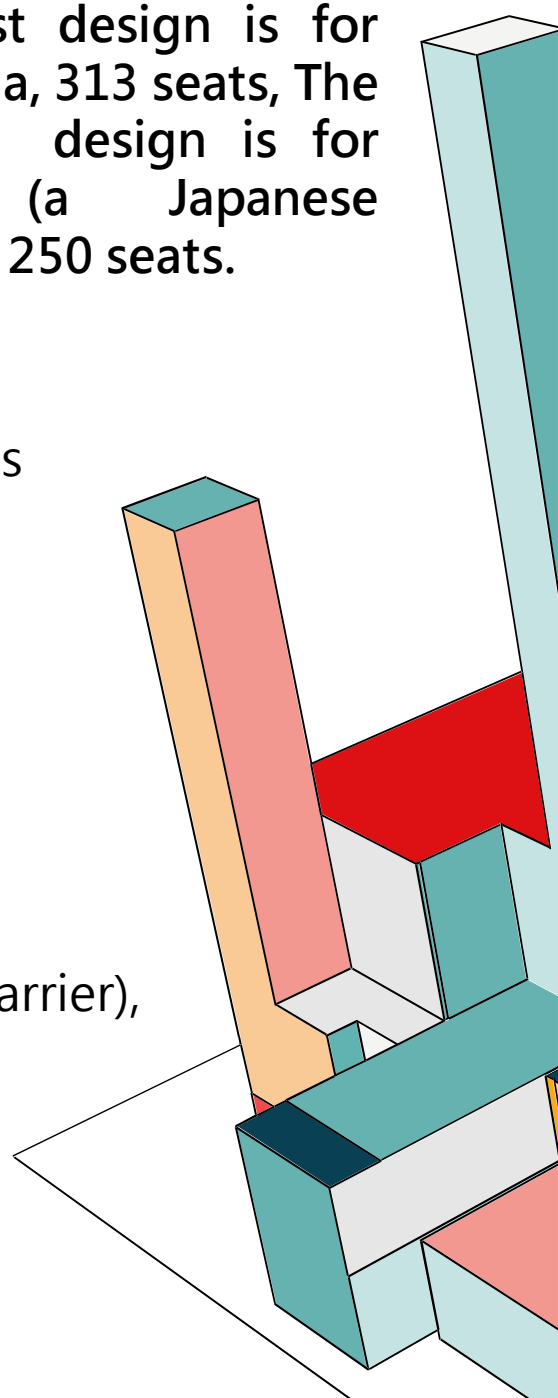
The first design is for Air China, 313 seats, The second design is for ANA (a Japanese carrier), 250 seats.

Air China, 313 seats

BOEING 777-300ER (Seats 250)



ANA (a Japanese carrier), 250 seats.



Architecture and internal design of the Boeing 777-300ER

- The same plane type that comes with same subtype can have multiple designs with similar predefined architecture.
- Previous Figures showed the architecture and internal design of the Boeing 777-300ER that are being built for the favor of three different Airline customers.
- The first design is for Air China, 313 seats, The second design is for ANA (a Japanese carrier), 250 seats. The blueprint includes most of the **Architecture and Design** specifications that are needed to enable plane developers to develop the plane.
- The development, or build activity itself that ends with built plane is called **product Development** activity (the fourth main activity in the production process). This activity ends with producing the plane.

- This plane couldn’ t be delivered to the customer without being tested aggressively. Its engines should be tested under the expected load of range, and weight. This activity is called Testing.
- At the end, the plane should be prepared to be deployed at the customer’ s predefined airport.
- To do this, the plane should be painted with the theme, and logo of the customer Airline.
- Finally, a crew flies by the plane to be deployed into the customer’ s airport. This activity is called Product Deployment.

Each SW development project includes those main six activities.

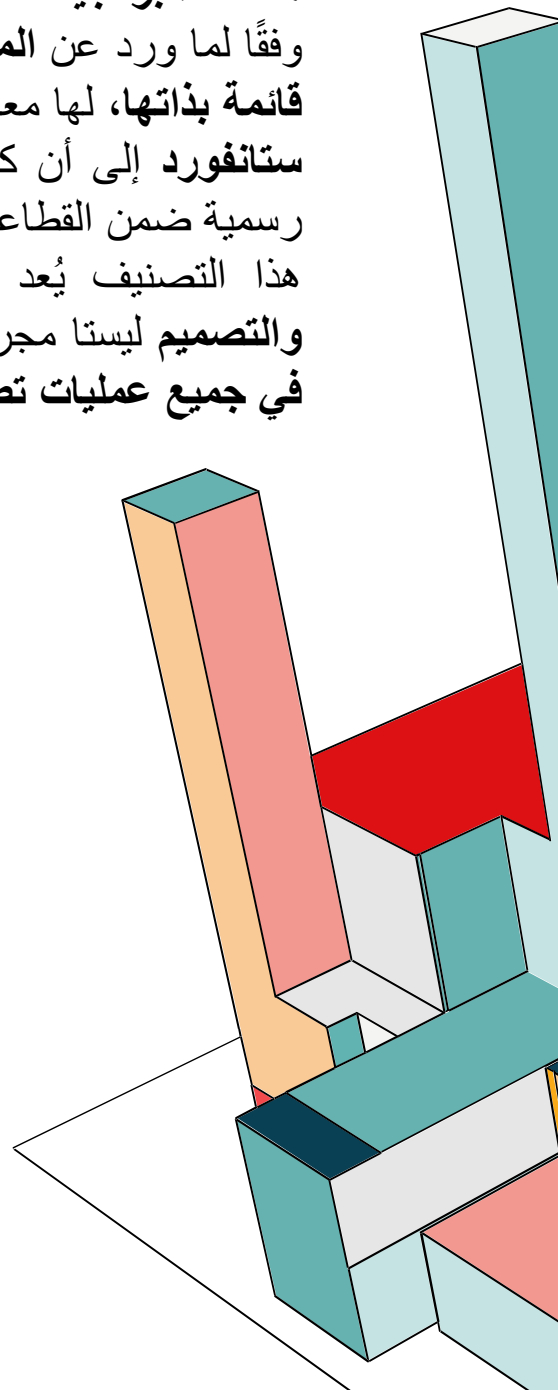
SO, IS SOFTWARE DEVELOPMENT IS A REAL INDUSTRY?

هندسة البرمجيات كصناعة متكاملة

وفقًا لما ورد عن **المنظمة الدولية للمعايير** (ISO)، يُعد تطوير البرمجيات **صناعة قائمة بذاتها**، لها معاييرها وممارساتها المهنية. كما تشير دراسة أُجريت في **جامعة ستانفورد** إلى أن كل من **الولايات المتحدة واليابان** تعتبران البرمجيات صناعة رسمية ضمن القطاعات الاقتصادية.

هذا التصنيف يُعد أمرًا جوهريًا، لأنه يعزز القناعة بأن **الهندسة المعمارية والتصميم** ليستا مجرد أنشطة تقنية في تطوير البرمجيات، بل هما **أنشطة أساسية في جميع عمليات تطوير المنتجات**، سواء كانت برمجية أو صناعية أو إلكترونية.

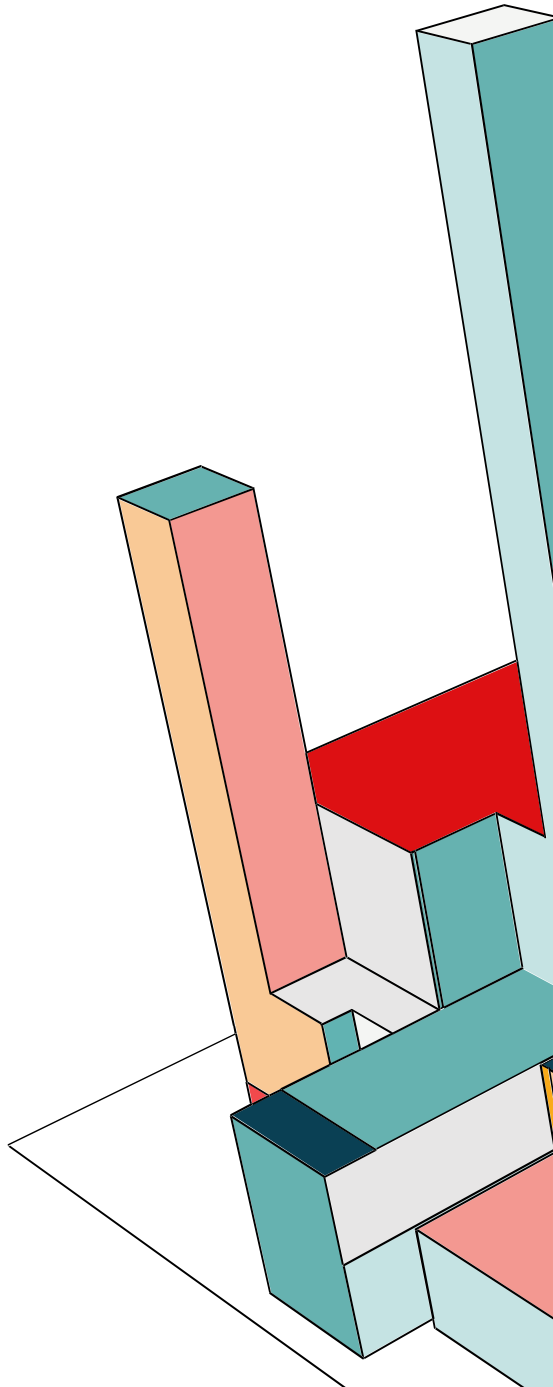
- According to International Organization for Standardization (**ISO**), software development is an industry.
- According to a study that has been conducted by Stanford University; SW is considered as an industry by US, and Japan.
- This mapping is important to grown the belief that architecture and design are core activities not in SW only, but also in any product development process.



LIKENESSES FROM OUTSIDE THE SOFTWARE WORLD

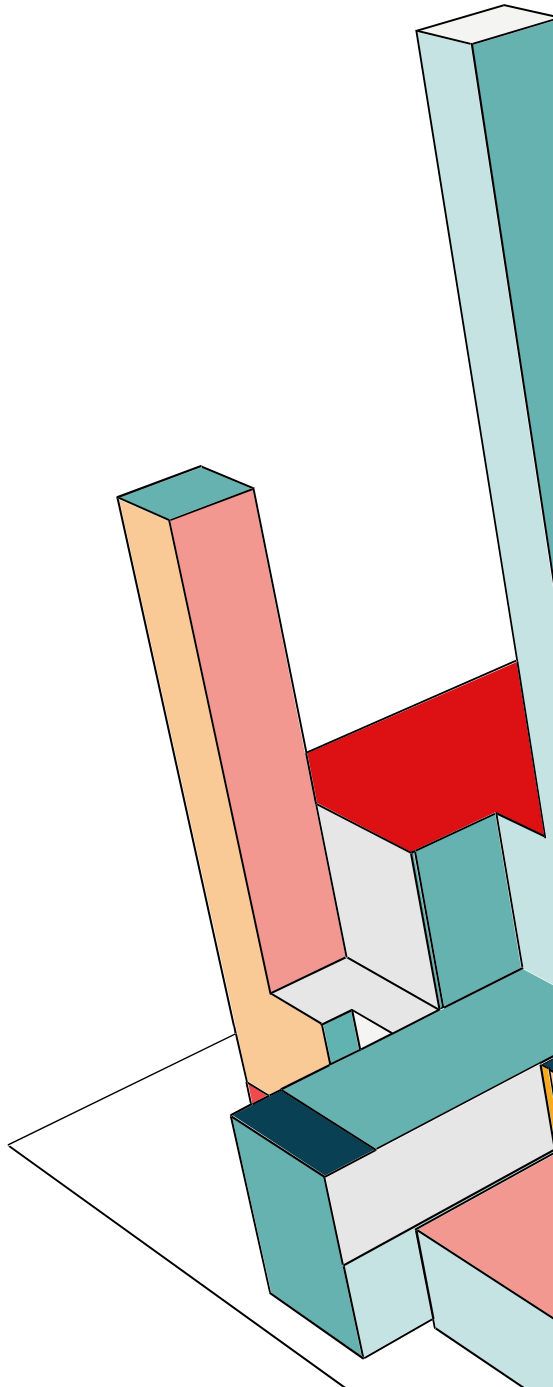
- Everything we build has a structure,
- The more we invest in building a product, the harder it becomes to change its structure.
- The thing is the structure of our system describes:
  - - The intent of our product .
  - - The qualities of the product.

كل منتج يتم تطويره يمتلك بنية داخلية تحدد طريقة عمله وتفاعله مع البيئة المحيطة. ومع تزايد الاستثمار في تطوير المنتج، تصبح عملية تعديل هذه البنية أكثر تعقيدًا وتكلفة. إن بنية النظام لا تعكس فقط الشكل الخارجي، بل تمثل نية التصميم والصفات الجوهرية للمنتج، مثل الأداء، القابلية للتوسع، الموثوقية، والكفاءة. وبالتالي، فإن فهم البنية وتوثيقها يعد خطوة أساسية في أي عملية تطوير منتج ناجحة.



محاور المحاضرة

- مقدمة ونظرة عامة عن مراحل تطوير النظم Software development Cycle
- **مرحلة التصميم أهميتها وترتيبها في دورة تطوير البرمجيات.** Importance of Design
  - عمليات التصميم والهيكلة
- أهمية تحليل وجمع المتطلبات وعلاقتها بمرحلة التصميم والهيكلة
  - المتطلبات الوظيفية
  - المتطلبات الغير وظيفية
  - قيود النظام
- الملاحق
  - المراجع
  - واجب



SOFTWARE ARCHITECTURE DEFINITION

"the software architecture of a system is a high-level description of the system structure, it's different components, and how those components communicate with each other to fulfill the system's requirements and constraints."

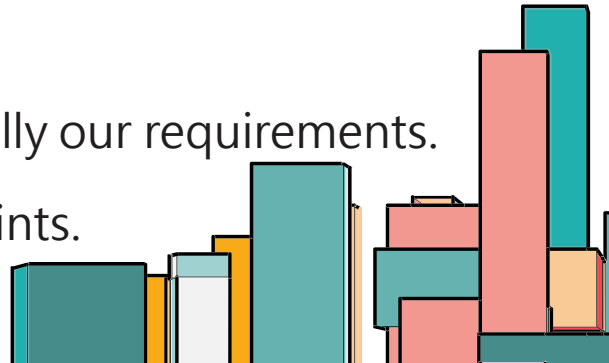
المعمارية المجردة للنظام تصورًا عالي المستوى يُظهر المكونات الأساسية التي تؤدي الوظائف المطلوبة، مع إخفاء التفاصيل التنفيذية الداخلية. يتم تعريف هذه المكونات بناءً على **سلوكها** و**واجهات البرمجة** APIs التي تتيح التفاعل بينها وبين الأنظمة الأخرى.

قد تكون بعض هذه المكونات أنظمة فرعية معقدة بحد ذاتها، تمتلك مخططات معمارية مستقلة.

تتعاون هذه المكونات لتحقيق ما يجب أن يقوم به النظام، أي **المتطلبات الوظيفية** المحددة مسبقًا.

وفي المقابل، يُضبط سلوك النظام بحيث **لا يقوم بما لا ينبغي عليه فعله**، وهو ما يُعرف بـ **قيود النظام**، مثل القيود الأمنية، الزمنية، أو التقنية.

- An abstraction that shows us the important components while hiding the implementation details.
- The components defined by their behavior and APIs.
- The components may themselves be complex systems with their own software architectural diagrams.
- The components are coming together to do what the system must do, which is basically our requirements.
- the system does not do what it shouldn't do, which is described in the system constraints.



LEVELS OF ABSTRACTION

Software architecture can have many different levels of abstraction:

- Classes,
- modules, packages, or libraries.
- Services (processes or groups of processes)

مستويات التجريد في المعمارية البرمجية

تُعد المعمارية البرمجية بنية متعددة الطبقات، يمكن تمثيلها عبر مستويات مختلفة من التجريد (Abstraction)، وذلك حسب زاوية النظر ومرحلة التطوير .هذه المستويات تشمل:

الفئات (Classes)

تمثل الوحدات الأساسية في البرمجة الكائنية، وتُستخدم لتحديد الكيانات وسلوكها داخل النظام.

الوحدات، الحزم، أو المكتبات (Modules, Packages, Libraries)

تُستخدم لتنظيم الكود في مجموعات منطقية قابلة لإعادة الاستخدام، وتُسهل الفصل بين المسؤوليات.

الخدمات (Services)

تشير إلى العمليات أو مجموعات العمليات التي تُنفذ وظائف محددة داخل النظام، وقد تكون مستقلة أو موزعة، مثل الخدمات المصغرة (Microservices)أو الخدمات السحابية.

IMPORTANCE OF DESIGN & ARCHITECTURE

Benefits :

If we get the architecture right we can:

a. Go from a small startup to a multi-billion dollar company.
b. Making a positive impact on millions of people.

Distributed, multi-service approach allows us to architect systems that can:

- ✓ handle large amounts of requests.
- ✓ process, and store very large amounts of data.
- ✓ serve many users every day.

Risks :

Not doing a good job at the design phase can:

- Waste months of engineering time.
- Building a system that doesn't meet our requirements.
  - Restructuring a system that was not architected correctly is very hard and expensive.
  - the stakes here are high.

ORDER OF ARCHITECTURE & DESIGN WITHIN THE SW DEVELOPMENT LIFECYCLE

- Most of the required information by Architect and Designer are Requirements that come from Analysis, and scoping constraints that come from Planning.
- In waterfall process model, Architecture, and design comes after Analysis.
- In agile iterative based processes, some architectural and design questions may lead to getting back to analysis thus, changing the final requirements upon received answers.

العلاقة بين التحليل والتصميم المعماري في نماذج العمليات البرمجية

تعتمد معظم المعلومات التي يحتاجها **المهندس المعماري والمصمم البرمجي** على **المتطلبات الوظيفية وغير الوظيفية** المستخلصة من مرحلة **التحليل**، بالإضافة إلى **قيود النطاق** التي يتم تحديدها خلال مرحلة **التخطيط**.

في نموذج الشلال التقليدي Waterfall Model، تأتي **مرحلة التصميم المعماري** بعد الانتهاء من التحليل، حيث تُبنى المعمارية على متطلبات ثابتة ومحددة مسبقًا.

أما في النماذج التكرارية المرنة مثل **المنهجية الرشيقة** Agile ، فإن بعض الأسئلة المعمارية أو التصميمية قد تؤدي إلى **العودة إلى مرحلة التحليل**، مما ينتج عنه **تعديل في المتطلبات النهائية** بناءً على الإجابات والمعطيات الجديدة التي تظهر أثناء التطوير.

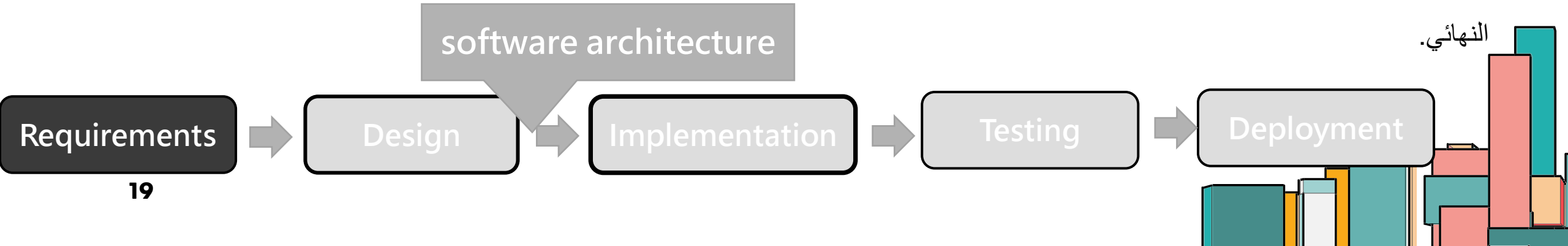
ORDER OF ARCHITECTURE & DESIGN WITHIN THE SW DEVELOPMENT LIFECYCLE

- Notes:
- Understanding customer’ s requirements is a key factor for having any good design.
- software architecture is the output of the design phase and the input to our systems implementation.

ملاحظات أساسية في التصميم المعماري للبرمجيات

1.**فهم متطلبات العميل هو العامل الأساسي للحصول على تصميم ناجح.**
إن دقة واستيعاب المتطلبات الوظيفية وغير الوظيفية منذ المراحل الأولى من التحليل، تُعد حجر الأساس لأي تصميم معماري فعال يلبي احتياجات المستخدمين ويحقق أهداف النظام.

2.**المعمارية البرمجية هي ناتج مرحلة التصميم، ومدخل أساسي لمرحلة تنفيذ النظام.** بعد الانتهاء من تحليل المتطلبات وتحديد القيود، يتم بناء المعمارية كنموذج هيكلي يوجه عملية التطوير، ويضمن تنظيم المكونات، وتوزيع المسؤوليات، وتحقيق الجودة المطلوبة في المنتج النهائي.



SOFTWARE ARCHITECTURE PROCESS

- Software architects spend a great deal of time working with software requirements.
  - ✓ Even after requirements are specified, software architects find themselves going back and forth between requirements and design.
  - ✓ In some cases, architects are completely immersed in the requirements phase, playing a key role in the specification of requirements.

دور المهندس المعماري في التعامل مع المتطلبات البرمجية

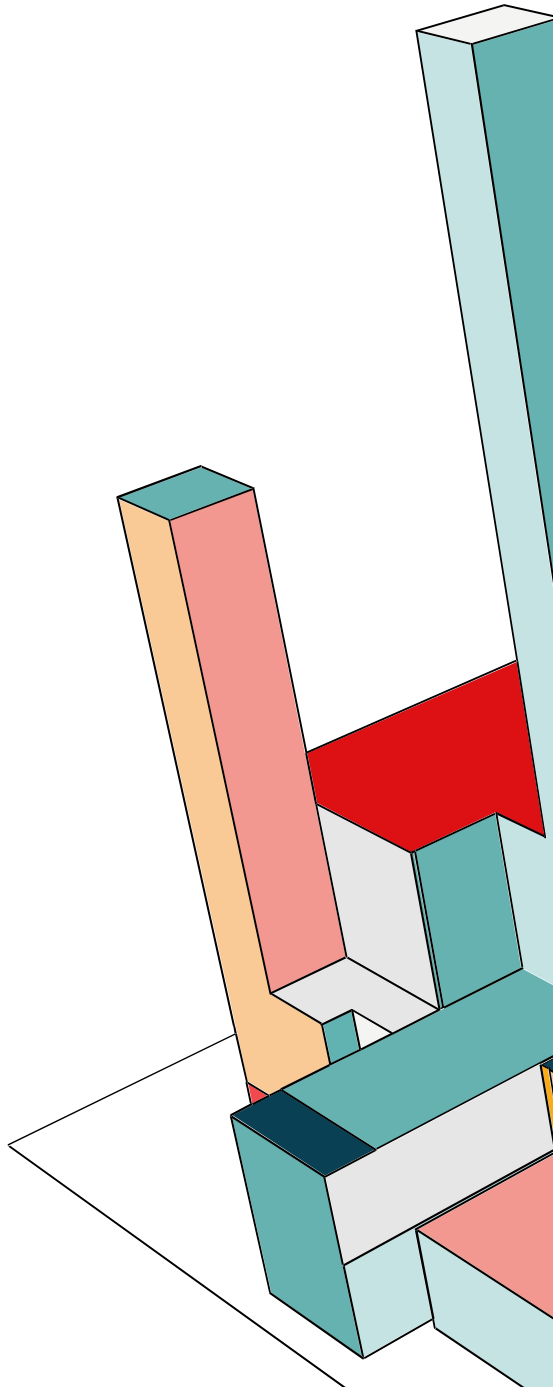
يقضي مهندسو البرمجيات المعماريون وقتًا كبيرًا في **التفاعل مع المتطلبات البرمجية**، نظرًا لأهميتها في تشكيل التصميم المعماري للنظام.

حتى بعد تحديد المتطلبات رسميًا، غالبًا ما يجد المهندس المعماري نفسه في حالة **تفاعل مستمر بين المتطلبات والتصميم**، حيث تؤثر القرارات التصميمية على فهم المتطلبات، وقد تستدعي مراجعتها أو تعديلها.

وفي بعض الحالات، يكون المهندس المعماري **منخرطًا بشكل كامل في مرحلة تحليل المتطلبات**، ويلعب دورًا محوريًا في **صياغة وتحديد المتطلبات** بالتعاون مع أصحاب المصلحة، لضمان توافقها مع أهداف النظام وإمكاناته التقنية.

محاور المحاضرة

- مقدمة ونظرة عامة عن مراحل تطوير النظم Software development Cycle
- مرحلة التصميم أهميتها وترتيبها في دورة تطوير البرمجيات. Importance of Design
  - **عمليات التصميم والهيكلة**
- أهمية تحليل وجمع المتطلبات وعلاقتها بمرحلة التصميم والهيكلة
  - المتطلبات الوظيفية
  - المتطلبات الغير وظيفية
  - قيود النظام
- الملاحق
  - المراجع
  - واجب



SOFTWARE ARCHITECTURE PROCESS

- On a larger scale, the process for creating software architectures can be executed using the following tasks:

1) Understand and evaluate requirements
2) Design the architecture
3) Evaluate the architecture
4) Document the architecture
5) Monitor and control implementation

تُنفذ عملية تصميم المعمارية البرمجية وفقًا لمجموعة من المهام:

1.**فهم وتقييم المتطلبات** تحليل المتطلبات الوظيفية وغير الوظيفية للنظام، وتحديد القيود التقنية والتنظيمية لضمان توافق التصميم مع أهداف المشروع.

2.**تصميم المعمارية** بناء الهيكل العام للنظام، وتحديد المكونات الأساسية، وطرق تفاعلها، واختيار الأنماط المعمارية المناسبة (مثل الطبقي، الخدمات المصغرة، SOA).

3.**تقييم المعمارية** مراجعة التصميم المعماري من خلال تحليل السيناريوهات، واختبار مدى تحقيقه للمتطلبات، وتحديد نقاط القوة والضعف باستخدام منهجيات مثل ATAM.

4.**توثيق المعمارية** إعداد وثائق رسمية تشمل المخططات، القرارات التصميمية، واجهات المكونات، وتبرير الاختيارات المعمارية باستخدام نماذج مثل C4أو arc42.

5.**مراقبة وتنفيذ المعمارية** متابعة تنفيذ النظام وفقًا للتصميم المعماري، وضمان الالتزام بالمواصفات، ومعالجة الانحرافات أو التعديلات أثناء التطوير.

MODELING AND DOCUMENTING

Without documenting the decisions of architecture and design we will be risking lots of aspect including but not limited to:

- The implementation.
- The maintainability.

  - For OO, UML is the documentation standard.
  - UML is an ISO standard # ISO/IEC 19505

أهمية توثيق القرارات المعمارية والتصميمية

إن **عدم توثيق القرارات المتعلقة بالتصميم والمعمارية البرمجية** يُعرض المشروع لمخاطر متعددة، تشمل - ولكن لا تقتصر على:

- **تنفيذ غير دقيق للنظام** نتيجة غياب المرجعية التصميمية.
- **صعوبة في الصيانة والتطوير المستقبلي** بسبب غموض الهيكل المعماري.
- **ضعف في التواصل بين أعضاء الفريق** نتيجة غياب التوثيق الرسمي.

في سياق **البرمجة الكائنية التوجه** Object-Oriented ، يُعد UML **لغة النمذجة الموحدة** المعيار المعتمد لتوثيق التصميم، حيث يوفر مجموعة من المخططات التي توضح البنية، السلوك، والتفاعلات داخل النظام.

وتجدر الإشارة إلى أن UML **هو معيار دولي رسمي** صادر عن المنظمة الدولية للمعايير تحت الرقم: ISO/IEC 19505

CHALLENGES OF SOFTWARE ARCHITECTURE

We cannot prove Software Architecture to be either:

- Correct .
- Optimal.

What we can do to guarantee success is follow:

- ✓ Methodical design process
- ✓ Architectural patterns
- ✓ Best practices.

تحديات في إثبات صحة المعمارية البرمجية وضمان نجاحها

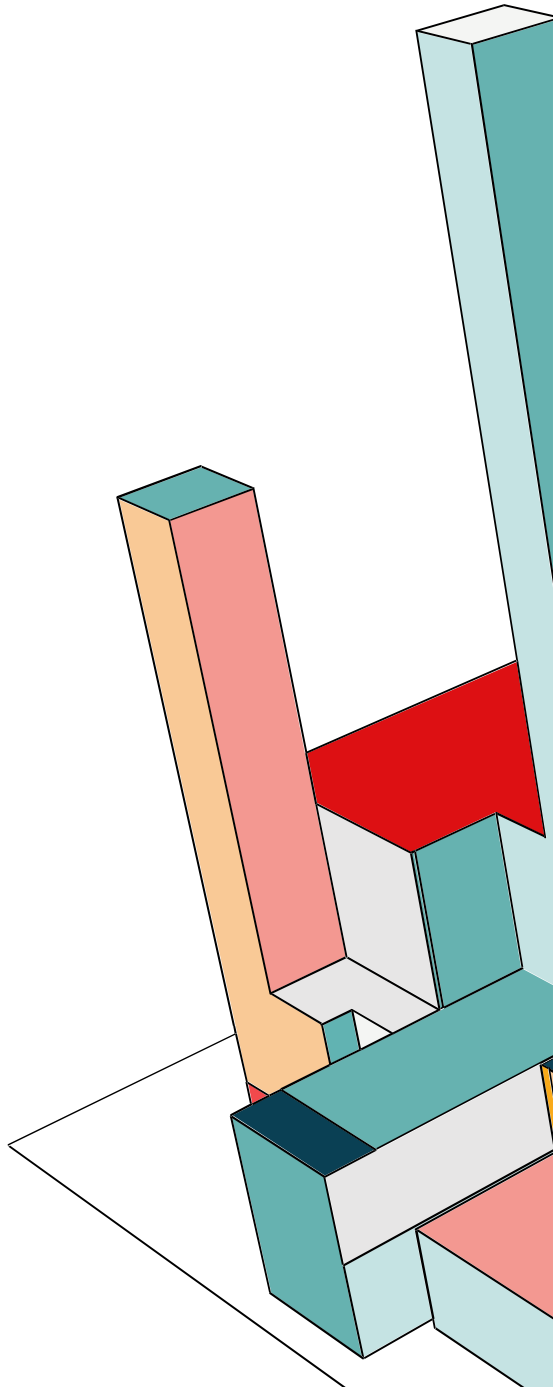
لا يمكننا **إثبات أن المعمارية البرمجية صحيحة أو مثالية بشكل مطلق**، وذلك بسبب تعقيد الأنظمة وتعدد المتغيرات التي تؤثر على الأداء والملاءمة.

ومع ذلك، يمكننا **تعزيز فرص نجاح النظام** من خلال اتباع منهجيات مدروسة تشمل:

- ✓ **عملية تصميم منهجية** تعتمد على خطوات واضحة تبدأ بفهم المتطلبات، مرورًا بالنمذجة، وانتهاءً بالتقييم والتوثيق.
- ✓ **الأنماط المعمارية المعتمدة** مثل النمط الطبقي، الخدمات المصغرة، أو النمط الموجه بالأحداث، والتي أثبتت فعاليتها في حالات استخدام متعددة.
- ✓ **أفضل الممارسات الهندسية** مثل الفصل بين المسؤوليات، تقليل الترابط، تعزيز القابلية للاختبار، وتوثيق القرارات التصميمية

محاور المحاضرة

- مقدمة ونظرة عامة عن مراحل تطوير النظم Software development Cycle
- مرحلة التصميم أهميتها وترتيبها في دورة تطوير البرمجيات. Importance of Design
  - عمليات التصميم والهيكلة
- **أهمية تحليل وجمع المتطلبات وعلاقتها بمرحلة التصميم والهيكلة**
  - المتطلبات الوظيفية
  - المتطلبات الغير وظيفية
  - قيود النظام
- الملاحق
  - المراجع
  - واجب



SOFTWARE DEVELOPMENT CYCLE

✓Similar to the design phase, the requirements phase can be broken down into well defined activities

Software Engineering Life-cycle Phase	Requirements →	Design →	Implementation
Phase Activities	Elicitation Analysis Specification Validation	Architecture Detailed Design	

الأنشطة الأساسية في مرحلة المتطلبات:

1. **جمع المتطلبات** Requirements Elicitation استخدام أدوات مثل المقابلات، الاستبيانات، وورش العمل لفهم احتياجات المستخدمين وأصحاب المصلحة.
2. **تحليل المتطلبات** Requirements Analysis تصنيف المتطلبات إلى وظيفية وغير وظيفية، وتحديد التناقضات أو الثغرات المحتملة.
3. **توثيق المتطلبات** Requirements Specification صياغة المتطلبات بشكل رسمي باستخدام نماذج مثل Software Requirements Specification SRS .
4. **التحقق من المتطلبات** Requirements Validation التأكد من أن المتطلبات دقيقة، قابلة للتحقيق، ومفهومة من قبل جميع الأطراف.
5. **إدارة المتطلبات** Requirements Management تتبع التغييرات، وضمان توافق المتطلبات مع مراحل التطوير والتصميم.

SYSTEM REQUIREMENTS AND ARCHITECTURAL DRIVERS

What are system Requirements ?

- ***Requirements*** : Formal description of what we need to build.
- large scale system requirements are a few differences from the usual requirements we typically get for implementing:
- Method
- Algorithm
- Class

المتطلبات Requirements

هي **وصف رسمي لما يجب بناؤه،** وتشكل الأساس الذي يُبنى عليه التصميم والتنفيذ في أي مشروع برمجي.

في الأنظمة البرمجية التقليدية، غالبًا ما تركز المتطلبات على عناصر محددة مثل:

- **الطرق** Methods وظائف محددة داخل الكائنات.
- **الخوارزميات** Algorithms خطوات منطقية لحل مشكلة معينة.
- **الفئات** Classes هياكل برمجية تمثل كيانات النظام.

أما في **الأنظمة واسعة النطاق** Large-scale systems ، فإن المتطلبات تختلف من حيث:

طبيعة المتطلبات: تكون أكثر **تجريدًا وتعقيدًا،** وتشمل سلوك النظام ككل وليس مجرد تفاصيل تنفيذية.

- تتضمن **متطلبات غير وظيفية** مثل الأداء، الأمان، القابلية للتوسع، والاعتمادية.

مستوى التفاعل: تشمل **تفاعل النظام مع أنظمة خارجية،** مستخدمين متعددين، وبيئات تشغيل متنوعة.

- تتطلب **تنسيقًا بين فرق متعددة،** مما يجعل التوثيق والتواصل أكثر أهمية.

التأثير على التصميم: تؤثر مباشرة على **اختيار المعمارية،** مثل استخدام الخدمات المصغرة أو النشر السحابي.

- تتطلب **تحليلًا معمقًا للقيود** مثل الزمن، الموارد، والتكامل مع أنظمة أخرى.

HIGH LEVEL OF AMBIGUITY

- *System Design has high level of ambiguity*
  - *Two reasons:*
  - ✓ *The person providing the requirements is often not an engineer and may even be not vey technical.*
  - ✓ *Getting the requirements is a part of the solution*
    - ❖ *The client doesn’ t always know what they need*
    - ❖ *The client generally knows only what problem they need solved*

- **غموض التصميم المعماري في الأنظمة البرمجية**
  - تتسم مرحلة **تصميم النظام** بدرجة عالية من **الغموض وعدم اليقين**، وذلك لأسباب متعددة، من أبرزها:
  - ✓ **عدم تخصص مقدم المتطلبات** غالبًا ما يكون الشخص الذي يحدد متطلبات النظام **ليس مهندسًا**، وقد لا يمتلك خلفية تقنية كافية، مما يؤدي إلى صياغة غير دقيقة أو غير مكتملة للمتطلبات.
  - ✓ **استخلاص المتطلبات جزء من الحل نفسه** في كثير من الحالات، لا يكون لدى العميل تصور واضح لما يحتاجه فعليًا، بل يعرف فقط **المشكلة التي يريد حلها**. وبالتالي، فإن عملية تحديد المتطلبات تصبح جزءًا من عملية بناء الحل، وليس مجرد خطوة مسبقة

IMPORTANCE OF GATHERING REQUIREMENTS

- *What happens if we don't get the requirements right?*
  - *We can simply build something and then fix it*
  - *Seemingly there's no cost of materials in software so changes should be cheap??*

- Large *scale systems are big projects that cannot be easily changed*
  - *Many engineers are involved*
  - *Many months of engineering work*
  - *Hardware and Software costs*
  - *Contracts include financial obligations*
  - *Reputation and brand*

⚠️ **ماذا يحدث إذا لم نحدد المتطلبات بشكل صحيح؟**

قد يبدو للوهلة الأولى أن **تعديل البرمجيات أمر سهل**، نظرًا لغياب تكلفة المواد الفيزيائية، مما يُغري البعض بالاعتقاد أنه يمكن **بناء أي شيء ثم إصلاحه لاحقًا**.

لكن هذا التصور **غير دقيق**، خاصة عند التعامل مع **أنظمة واسعة النطاق**، حيث تكون التغييرات مكلفة ومعقدة للأسباب التالية:

أسباب تجعل تعديل الأنظمة الكبيرة أمرًا صعبًا:

- **عدد كبير من المهندسين** يعملون على النظام، مما يجعل التنسيق وإعادة التوجيه أمرًا معقدًا.
- **أشهر من العمل الهندسي** قد تضيع إذا تم تعديل المتطلبات بعد التنفيذ.
- **تكاليف مادية** تشمل الأجهزة، التراخيص، البنية التحتية، وخدمات التشغيل.
- **عقود رسمية** تتضمن التزامات مالية وزمنية، يصعب تعديلها دون تبعات قانونية.
- **سمعة الشركة والعلامة التجارية** قد تتأثر سلبًا إذا فشل النظام أو لم يلبِّ توقعات العملاء.

TYPES OF REQUIREMENTS

- *Features of the System*
  - *Functional requirements*
- *Quality Attributes*
  - *Non-Functional requirements*
- *System Constraints*
  - *Limitations and boundaries*

وظائف النظام Features of the System
تمثل **المتطلبات الوظيفية** Functional Requirements، وهي ما يجب أن يقوم به النظام من وظائف وسلوكيات ملموسة، مثل:
تسجيل الدخول والخروج
معالجة الطلبات
توليد التقارير
إرسال الإشعارات

سمات الجودة Quality Attributes
تمثل **المتطلبات غير الوظيفية** Non-Functional Requirements ،
وهي خصائص النظام التي تؤثر على طريقة عمله وليس على الوظائف نفسها، مثل:
الأداء Performance
الأمان Security
القابلية للتوسع Scalability
الاعتمادية Reliability
سهولة الاستخدام Usability

القيود النظامية System Constraints
تشير إلى **القيود والحدود المفروضة** على النظام، والتي تُحدد إطار العمل الممكن، مثل:
استخدام لغة برمجة معينة
التوافق مع نظام تشغيل محدد
•الالتزام بلوائح قانونية مثل GDPRأو HIPAA
•قيود الميزانية أو الجدول الزمني

TYPES OF REQUIREMENTS

- *Features / Functional Requirements*

  Describe the system behavior - what "the system must do"

  Easily tied to the objective of our system

- *Functional requirements do not determine its system architecture*

  Generally, any architecture can achieve any feature

من المفاهيم الأساسية في هندسة البرمجيات أن:

المتطلبات الوظيفية لا تحدد المعمارية البرمجية بشكل مباشر.

بمعنى أن أي وظيفة أو ميزة يمكن تنفيذها باستخدام **أنماط معمارية متعددة**، مثل النمط الأحادي Monolithic أو الخدمات المصغرة Microservices ، أو حتى المعمارية الموجهة نحو الأحداث Event-Driven Architecture

لماذا لا تحدد المتطلبات الوظيفية المعمارية؟

- لأن **الوظائف** تصف *ما يجب أن يفعله النظام*، بينما **المعمارية** تحدد *كيف يتم تنظيم النظام لتحقيق تلك الوظائف*.
- يمكن تنفيذ نفس الوظيفة (مثل تسجيل الدخول أو معالجة الطلبات) باستخدام هياكل مختلفة تمامًا من حيث التوزيع، الأداء، والتوسع.

ما الذي يحدد المعمارية فعليًا؟

- **المتطلبات غير الوظيفية (سمات الجودة)** مثل الأداء، الأمان، القابلية للتوسع، والاعتمادية.
- **القيود التقنية والتجارية والتنظيمية** التي تفرض حدودًا على الحلول الممكنة.
- **البيئة التشغيلية** وعدد المستخدمين المتوقعين وحجم البيانات.

TYPES OF REQUIREMENTS

Quality Attributes / Non-Functional Requirements System properties that "the system much have"

Examples:

- ❑ *Scalability*
- ❑ *Availability*
- ❑ *Reliability*
- ❑ *Security*
- ❑ *Performance*

Quality attributes and Software Architecture

• The quality attributes dictate the software architecture of our system

SYSTEM CONSTRAINTS

"A system constraint is essentially a decision that was already either fully or partially made for us, restricting our degrees of freedom."

Instead of looking at a constraint as a choice that was taken away, we look at it as a decision that was already made

System Constraints are referred as pillars for software architecture because:

- They provide us with a solid starting point
- The rest of the system need to be designed around them

القيود النظامية كركائز للمعمارية البرمجية

يُنظر إلى **القيود النظامية** System Constraints على أنها **قرارات تم اتخاذها مسبقًا** – سواء بشكل كامل أو جزئي – مما يُقلل من مساحة الحرية المتاحة أثناء التصميم.

بدلًا من اعتبار القيود على أنها **خيارات مسلوبة**، يمكننا رؤيتها على أنها **أسس تم تحديدها مسبقًا**، تُساعد في توجيه عملية التصميم وتحديد المسارات الممكنة.

ولهذا السبب، تُعد القيود النظامية بمثابة **ركائز أساسية للمعمارية البرمجية**، لأنها:

- **توفر نقطة انطلاق واضحة وصلبة** تُحدد الإطار العام الذي يجب أن يعمل ضمنه النظام، مثل نوع المنصة، اللغة البرمجية، أو متطلبات الأمان.
- **تُشكل الأساس الذي يُبنى عليه باقي التصميم** يجب أن يتم تصميم باقي مكونات النظام بما يتوافق مع هذه القيود، لضمان التكامل والامتثال

SYSTEM CONSTRAINTS

Types of Constraints

There are three types of constraints:

- Technical constraints
- Business constraints
- Regulatory/legal constraints

أنواع القيود في هندسة البرمجيات

عند تصميم الأنظمة البرمجية، يواجه المهندسون مجموعة من **القيود (Constraints)** التي تُحدد إطار العمل وتؤثر على القرارات المعمارية. وتنقسم هذه القيود إلى ثلاث فئات رئيسية:

نوع القيد	الوصف	أمثلة واقعية
القيود التقنية	تحدد الأدوات والتقنيات التي يجب استخدامها أو تجنبها. تؤثر على التصميم والتنفيذ.	لغة برمجة محددة، نظام تشغيل معين، أداء مطلوب
القيود التجارية	ناتجة عن متطلبات العمل أو السوق، وتفرض حدودًا على الميزانية أو الجدول الزمني.	ميزانية محدودة، موعد تسليم صارم، توافق مع أنظمة قائمة
القيود القانونية والتنظيمية	تفرضها الجهات التشريعية، وتلزم النظام بالامتثال لقوانين الخصوصية والأمان.	GDPR في أوروبا، HIPAA في أمريكا

SYSTEM CONSTRAINTS

Technical Constraints

- Examples of technical constraints include:

  Being locked to a particular **hardware/cloud vendor**

  Having to use a particular **programming language**

  Having to use a particular **database or technology**

  Having to support certain **platforms, browsers, or OS**

أمثلة على القيود التقنية:

- ❑ **الارتباط بمزود معين للأجهزة أو الخدمات السحابية** مثل الالتزام باستخدام AWS أو Azure فقط، مما يؤثر على البنية التحتية والتكامل.
- ❑ **استخدام لغة برمجة محددة** كأن يُطلب بناء النظام بلغة Javaأو Pythonفقط، مما يحد من استخدام تقنيات أخرى.
- ❑ **استخدام قاعدة بيانات أو تقنية معينة** مثل فرض استخدام Oracle أو MongoDB ، مما يؤثر على تصميم البيانات والتخزين.
- ❑ **دعم منصات أو متصفحات أو أنظمة تشغيل معينة** مثل ضرورة توافق النظام مع Windows وChrome فقط، مما يفرض قيودًا على تجربة المستخدم والتطوير

SYSTEM CONSTRAINTS

Technical Constraints

- Technical constraints may seem like they belong to implementation and not to software architecture
- In practice, they affect the decisions we make in the design phase and put restrictions on our architecture.

القيود التقنية وتأثيرها على المعمارية البرمجية

قد يبدو للوهلة الأولى أن **القيود التقنية** Technical Constraints تنتمي إلى مرحلة **التنفيذ البرمجي** فقط، مثل اختيار الأدوات أو كتابة الكود.

لكن في الواقع، هذه القيود **تلعب دورًا حاسمًا في مرحلة التصميم المعماري**، حيث تؤثر بشكل مباشر على القرارات التي يتخذها المهندس المعماري، وتُحدد حدود الحلول الممكنة.

أمثلة على القيود التقنية:

- الاعتماد على لغة برمجة معينة أو إطار عمل محدد.
- التوافق مع نظام تشغيل أو بيئة تشغيل معينة.
- قيود الأداء أو حجم الذاكرة المتاحة.
- استخدام قاعدة بيانات أو بروتوكول معين مفروض مسبقًا.

SYSTEM CONSTRAINTS

Technical Constraints Examples

- **Example 1:** If our company makes a decision to run on-premise data center, then:
  - ✓ All the cloud architectures and paradigms will become unavailable to us
- **Example 2:** If we have to support some older browsers or low-end mobile devices then:
  - o We have to adapt our architecture to support those platforms and their APIs
  - o Keep providing a different, more high-end experience for newer browsers or higher-end devices

أمثلة على القيود التقنية وتأثيرها على التصميم المعماري

المثال الأول: تشغيل النظام في مركز بيانات داخلي (On-Premise)

إذا قررت الشركة تشغيل النظام داخل مركز بيانات خاص بها، فإن هذا القرار يُعتبر قيدًا تقنيًا يؤدي إلى:

- **استبعاد المعماريات السحابية** مثل النشر على AWS أو Azure.
- **الاعتماد على البنية التحتية المحلية**، مما يحد من قابلية التوسع التلقائي أو استخدام خدمات سحابية جاهزة.
- **تعديل التصميم المعماري** ليتوافق مع بيئة التشغيل الداخلية، مثل إدارة الموارد يدويًا أو استخدام أدوات مراقبة محلية.

المثال الثاني: دعم متصفحات قديمة أو أجهزة منخفضة الأداء

إذا كان النظام مطالبًا بدعم متصفحات قديمة أو أجهزة محمولة ضعيفة، فإن ذلك يفرض:

- **تكييف المعمارية** لتعمل بكفاءة على تلك المنصات، مثل تقليل استخدام JavaScript الثقيل أو تقنيات حديثة غير مدعومة.
- **توفير تجربة مختلفة ومحسّنة** للمستخدمين على الأجهزة الحديثة، مما يتطلب تصميمًا مرنًا يدعم التكيف حسب المنصة.
- **زيادة التعقيد في التنفيذ** بسبب الحاجة إلى تعدد واجهات الاستخدام أو تقنيات التوافق.

SYSTEM CONSTRAINTS

Business Constraints

- As engineers, we make the right decisions and architectural choices from a technical perspective
- This forces us to make sacrifices in:
  - ➢Architecture
  - ➢Implementation

القيود التجارية وتأثيرها على التصميم المعماري

بصفتنا مهندسين، نسعى دائمًا لاتخاذ **القرارات التقنية المثلى** واختيار أفضل الحلول المعمارية من منظور هندسي بحت. لكن في الواقع العملي، هناك **قيود تجارية** (Business Constraints) تفرض نفسها على المشروع، مما يجبرنا أحيانًا على **تقديم تنازلات** في:

- 📐 **التصميم المعماري**: مثل تقليل عدد المكونات أو تبسيط البنية لتقليل التكلفة أو تسريع التسليم.
- 🛠️ **التنفيذ البرمجي**: مثل استخدام تقنيات أقل كفاءة لكنها مألوفة للفريق أو أرخص في التشغيل.

🔒 **أمثلة على القيود التجارية:**

- الميزانية المحدودة
- الجدول الزمني الضيق
- التوافق مع أنظمة موجودة مسبقًا
- متطلبات السوق أو العملاء
- قرارات إدارية أو تعاقدية

SYSTEM CONSTRAINTS

Business Constraints - Examples

- Limited budget or a strict deadline will make us have very different choices than if we had an unlimited budget and unlimited time
- Different software architectural patterns are based on suitability between small startups or bigger organizations.
- Usage of third-party services with their own APIs and architectural paradigms as part of our architecture
  - ✓ Using third-party shipping/billing providers for an online store
  - ✓ Integration of different banks/brokers/security/fraud detection services for an investing platform

أمثلة على القيود التجارية وتأثيرها على التصميم المعماري

القيود التجارية هي **عوامل غير تقنية** تُفرض من قبل بيئة العمل أو السوق، وتؤثر بشكل مباشر على القرارات المعمارية والتنفيذية في المشروع البرمجي.

أمثلة واقعية:

الميزانية المحدودة أو الجدول الزمني الصارم

تؤدي إلى اختيار حلول أبسط أو جاهزة بدلًا من بناء مكونات مخصصة.

قد يتم التضحية ببعض السمات غير الوظيفية مثل الأداء أو القابلية للتوسع.

اختلاف الأنماط المعمارية حسب حجم المؤسسة

الشركات الناشئة تميل إلى استخدام أنماط خفيفة مثل النمط الأحادي (Monolithic).

المؤسسات الكبرى تفضل أنماط أكثر تعقيدًا مثل الخدمات المصغرة (Microservices) أو SOA.

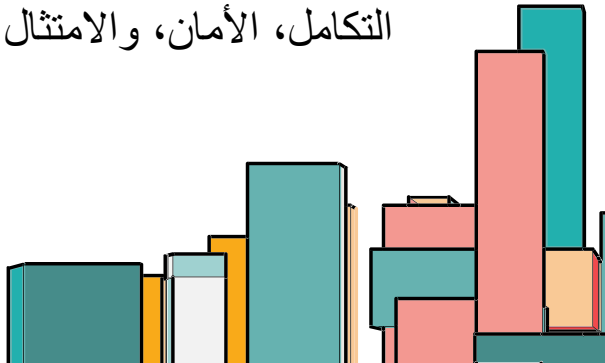
استخدام خدمات خارجية بواجهات برمجية خاصة (APIs)

مثل الاعتماد على مزودي الشحن أو الدفع الإلكتروني في المتاجر الإلكترونية.

يتطلب التصميم المعماري التكيف مع هذه الخدمات وقيودها التقنية.

دمج أنظمة مالية متعددة في منصة استثمارية

مثل البنوك، الوسطاء، أنظمة كشف الاحتيال، مما يفرض قيودًا على التكامل، الأمان، والامتثال التنظيمي



SYSTEM CONSTRAINTS

Regulatory/Legal Constraints

- Regulatory constraints may be:
  - Global
  - Specific to a region
- Examples:

✓ In the US, HIPAA (Health Insurance Portability and Accountability Act) places constraints on accessing patients' data

✓ In the European Union, GDPR (General Data Protection Regulation)

sets limitations on collecting, storing and sharing users' data

القيود التنظيمية والقانونية في تطوير البرمجيات

تُعد **القيود التنظيمية والقانونية (** Regulatory/Legal Constraints)من العوامل الأساسية التي يجب أخذها بعين الاعتبار أثناء تصميم وتطوير الأنظمة البرمجية، لما لها من تأثير مباشر على المعمارية، التنفيذ، والتشغيل.

هذه القيود قد تكون:

- **عالمية**: تنطبق على جميع الأنظمة بغض النظر عن موقعها، مثل المعايير الدولية للأمان أو الخصوصية.
- **إقليمية أو محلية**: تختلف حسب الدولة أو المنطقة، وتُفرض من قبل الجهات التشريعية المحلية.

أمثلة واقعية:

- **في الولايات المتحدة**: قانون) HIPAAقانون قابلية نقل وحماية معلومات التأمين الصحي) يفرض قيودًا صارمة على **الوصول إلى بيانات المرضى**، ويُلزم الأنظمة الصحية بتطبيق ضوابط خصوصية وأمان دقيقة.
- **في الاتحاد الأوروبي**: تنظيم) GDPRاللائحة العامة لحماية البيانات) يضع قيودًا على **جمع، تخزين، ومشاركة بيانات المستخدمين**، ويُلزم الشركات بالحصول على موافقة واضحة من المستخدمين، وتوفير آليات لحذف البيانات عند الطلب.

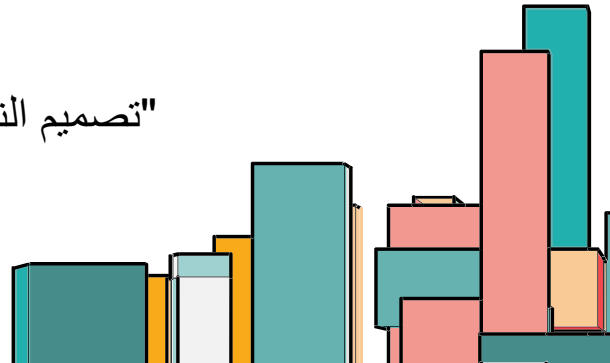
المحفزات المعمارية (Architectural Drivers)
المحفزات المعمارية هي العوامل الأساسية التي تؤثر بشكل مباشر على **تصميم المعمارية البرمجية**، وتُحدد شكل النظام، مكوناته، وآلية تفاعله مع البيئة المحيطة.
تُعد هذه المحفزات بمثابة **نقاط انطلاق استراتيجية** يجب على المهندس المعماري مراعاتها منذ المراحل الأولى للتصميم.

TYPES OF REQUIREMENTS

- **Features of the System**
  - o Functional requirements
- **Quality Attributes**
  - o Non-Functional requirements
- **System Constraints**
  - o Limitations and boundaries

} Architectural Drivers

"تصميم النظام البرمجي يتطلب فهمًا دقيقًا للوظائف المطلوبة، السمات النوعية، والقيود المفروضة، لضمان بناء نظام متكامل وفعّال."



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 1

المطلوب: يرجى من الطلبة إعداد وثيقة تعريف بالمشكلة (Problem Statement) تتناول ظاهرة إدمان الأطفال للهواتف الذكية في عمر مبكر، وذلك من منظور تقني وسلوكي. يجب أن تتضمن الوثيقة وصفًا دقيقًا للمشكلة، أسبابها، آثارها المحتملة، والفئة المستهدفة.

المتطلبات:

صياغة واضحة ومركزة لمشكلة الإدمان الرقمي لدى الأطفال.

تحديد الفئة العمرية المستهدفة (مثلاً: من 3 إلى 10 سنوات).

الإشارة إلى دور التطبيقات الرقمية في تعزيز أو تقليل هذا الإدمان.

لا تتجاوز الوثيقة صفحة واحدة أو صفحتين كحد أقصى.

الهدف: تمهيد لتحليل الحلول التقنية الممكنة وتصميم تطبيق موجه للأطفال يراعي الجوانب النفسية والسلوكية.

- الاستلام من صفحة المادة على الخاص.