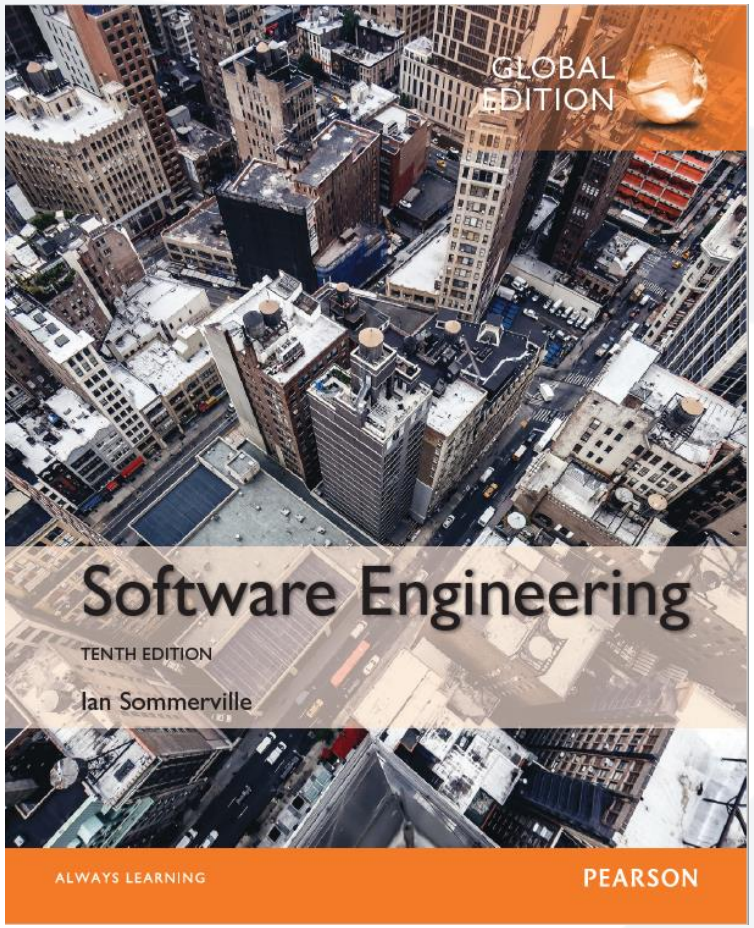
جَامعة
المَنارة
MANARA UNIVERSITY

Software Engineering -2-

Lecture -1-

Dr. Inas Laila

المرجع المعتمد



Sommerville, I. (2016). Software Engineering (10th ed.). ISBN:978-1-292-09613-1. Pearson Education.

معلومات إضافية ذات صلة:

https://iansommerville.com/software-engineering-book/static/about/

Complexity : The system is so complex that **no single** programmer can understand it anymore

هل تتشابه البرمجيات؟

ادى ذلك الى ظهور الفريق البرمجي

Software development requires good managers. The manager who can understand the psychology of the people and provide good leadership. After having a good manager, project is in safe hands. It is the responsibility of a manager to manage, motivate, guide and control the people of his/her team.

هل تتشابه البرمجيات ؟؟

بعض البرمجيات قد تؤدي الى مشاكل كارثية ان لم تعمل بشكل صحيح

هذه البرمجيات تتطلب تكلفة اضافية في مرحلة الاختبار

Some Software that does not work correctly can lead to many problems, including loss of money, and could even cause injury or death.

60% of software costs are development costs, 40% are testing costs

"31% of projects get cancelled before they are completed, 53% over-run their cost estimates by an average of 189% and for every 100 projects, there are 94 restarts" IBM Report

There are serious problems in the cost, timeliness, maintenance and quality of many software products

So what should we do?
How could we react?

Software Engineering: Definition

هندسة البرمجيات فرع من فروع الهندسة مرتبط بتطوير المنتجات البرمجية باستخدام مبادئ وتقنيات وإجراءات علمية محددة بدقة. وتنتج هندسة البرمجيات منتجاً برمجياً فعالاً وموثوقاً.

We use Software Engineering to build

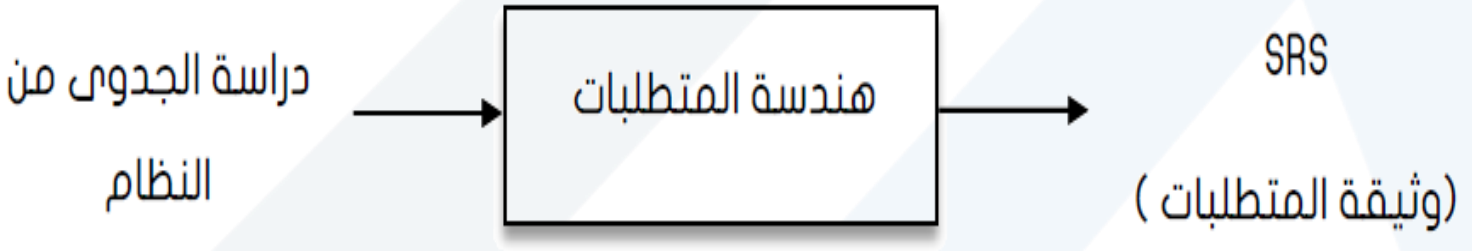
- a high quality maintainable software
- with a given budget
- before a given deadline (produce software on time)

درسنا في مقرر هندسة البرمجيات ١ النماذج التي يمكن اتباعها عند تطوير النظام البرمجي كنموذج الشلال والحلزوني والتزايدي والنماذج الرشيقة وغيرها وتحدثنا عن النشاطات التي يجب نمر بها عند استخدام هذه النماذج وترتيب ورودها

حيث تعمقنا بمرحلة التحليل

كان دخل مرحلة التحليل هو دراسة الجدوى (feasibility study) والتي درسنا فيها مدى الجدوى من المشروع وامكانية البدء به

أما خرجها فكان وثيقة توصيف المتطلبات (SRS)



و وثيقة ال SRS تلعب 3 أدوار و هي :

1- تكون بمثابة وثيقة لتوقيع العقد مع الزبون .

2- هذه الوثيقة تكون دخل لمرحلة التصميم و هي نقطة البداية التي يبدأ بها المصممون لتصميم النظام اعتماداً عليها .

3- أهم دور لها أنها مرجع أساسي في عمليات ال testing و قبول النظام أو بما نسميه (اختبار قبول النظام) أو acceptance testing .

حيث قسمنا المتطلبات الى:

1) ***Functional Requirement*** : (المتطلبات الوظيفية)

و نقصد بها الخدمات الفعلية التي يقدمها النظام (إجرائيات العمل) أي الوظائف الأساسية التي يقدمها النظام مثل (خدمة الحجز ، العرض ، التسجيل ... و غيرها) .

2) ***Nonfunctional Requirement*** : (المتطلبات الغير وظيفية)

و هي عبارة عن القيود على المتطلبات و هي تعد بمثابة متطلبات مثلاً:

1- **متطلبات تتعلق بالجودة :**

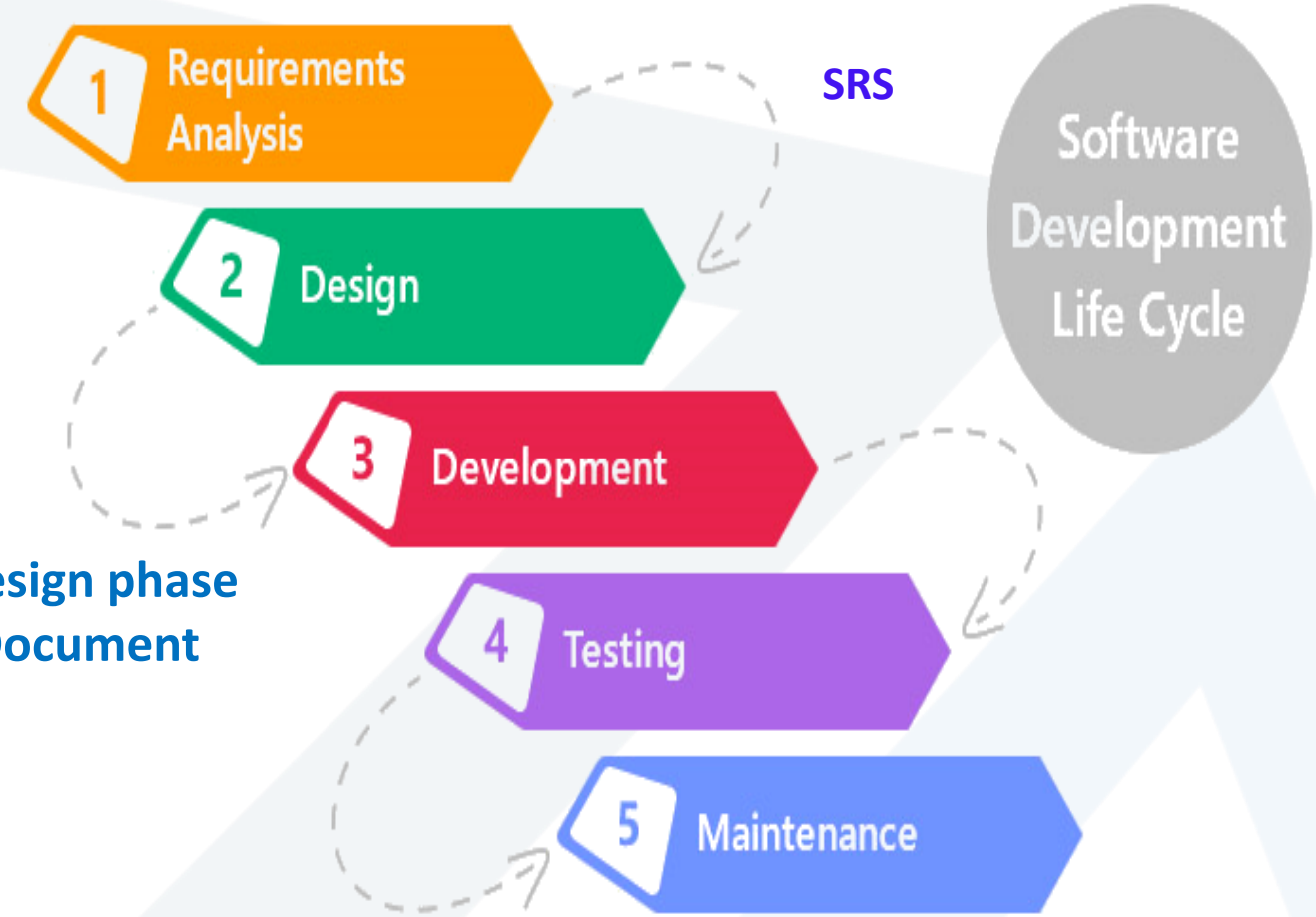
مثل ال security و ال reliability و performance و ال availability .

2- **متطلبات متعلقة بالكلفة (cost) :**

مثلاً زبون يطلب تطوير نظام تحت كلفة لا تتجاوز ال 100,000$ هذا يعد متطلب و هو قيد أيضاً .

3- **متطلبات متعلقة بالوقت (time) :**

أي إعطاء فترة محددة من الزمن لتطوير نظام معين .



The output of the design phase is Software Design Document (SDD).

بعد الانتهاء من مرحلة التحليل ننتقل لمرحلة التصميم , والفرق بين التحليل والتصميم هو التالي :

1) يُمَثل التحليل تَصوّر مبدئي مؤقت عن النظام (high-level representations) , أما التصميم فيمثل التصور التفصيلي عن الحل .
2) التصميم يهتم ببيئة النظام الخارجية أو ال system domain (مواصفات الجهاز الذي سيَشغل التطبيق , التوافق مع الأنظمة الأخرى , مواصفات الشبكة , ...) أما التحليل يُصور الحل تجريدياً دون مراعاة لل domain .
3) التصميم يهتم بتحقيق المتطلبات غير الوظيفية إضافة إلى المتطلبات الوظيفية , أما التحليل فيلقي نظرة مبدئية على المتطلبات الوظيفية فقط .
4) التصميم يجيب على السؤال التالي :

كيف يجب تحقيق النظام ؟

أما التحليل يجيب عن السؤال التالي :

ما الذي يفترض أن يقوم به النظام ؟

تشكل نتائج مرحلة التحليل نقطة البدء لمرحلة التصميم , بينما تشكل مرحلة التصميم نقطة البدء لمرحلة التنفيذ implementation .

مرحلة التصميم: هي مرحلة خلاقة، تعني الانتقال من فضاء المشكلة (problem domain) إلى فضاء الحل (solution Space) عن طريق هذه المرحلة حيث نحدد شكل النظام بحيث يلبي المتطلبات الوظيفية ويحترم المتطلبات غير الوظيفية.

تشمل مرحلتين:

1. التصميم المعماري: أول خطوة للتصميم دخلها هو وثيقة SRS مع المخططات التي تحويها ويدرس كيف ننظر للنظام ككل مكون من أنظمة جزئية متفاعلة مع بعضها البعض.
2. التصميم التفصيلي: يشرح التصميم التفصيلي لكل مكون من حيث الخوارزميات والواجهات وقاعدة البيانات.

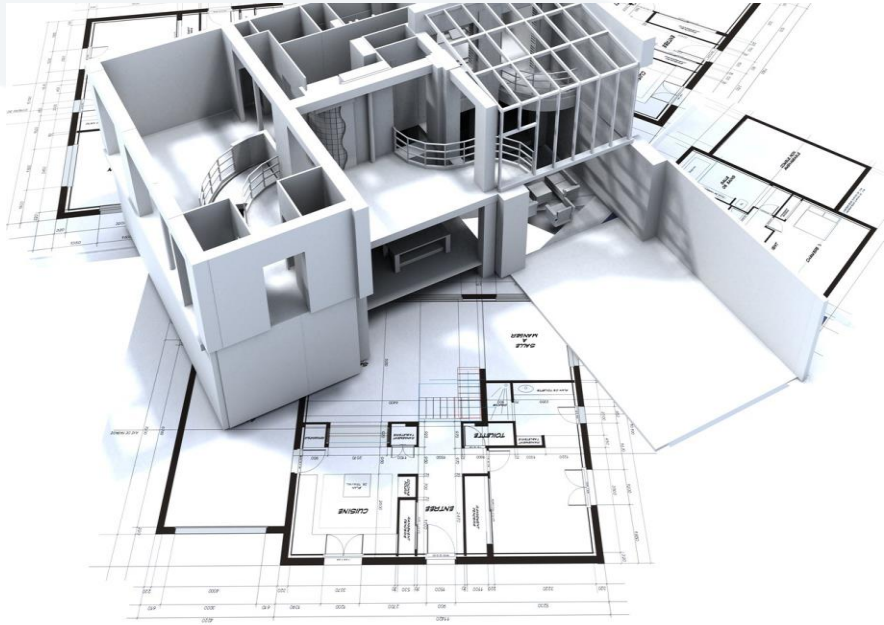
ناتج هذه المرحلة هو Design Document في هذه المحاضرة سندرس التصميم المعماري:

وهو يدرس هيكلية النظام (Structure)

التصميم المعماري في هندسة البرمجيات
Architectural Design in Software Engineering

التصميم المعماري في هندسة البرمجيات هو تصميم رفيع المستوى، يُعبَّر عنه بمخطط كتلي يُحدد بنية البرنامج.
يُحدد هذا المخطط بنية وخصائص المكونات، وأنماط تدفق البيانات، وكيفية تواصل هذه المكونات مع بعضها البعض لمشاركة البيانات.
ببساطة، تتضمن عملية التصميم المعماري تحديد المكونات، أي الأنظمة الفرعية التي تشكل النظام والتفاعل فيما بينها.
يساعد التصميم الجيد لبنية البرمجيات على تحديد الأداء، وقابلية الصيانة، والجودة، وقابلية التوسع.
الهدف من التصميم الجيد هو ضمان مرونة برمجياتك بما يكفي للتكيف مع ظهور متطلبات جديدة.
يتم الاسترشاد عند التصميم بالأنماط والأساليب المعمارية، التي توفر قوالب قابلة لإعادة الاستخدام لتنظيم المكونات وإدارة التفاعلات. تشمل الأنماط المعمارية الشائعة:

Layered Architecture, Client-Server, Microservices, and Event-Driven Architecture.....



التصميم المعماري للبرمجية يشبه التصميم المعماري للأبنية

Systems in the same domain often have **similar architectures** that reflect domain concepts.

في الطرق التقليدية كان النظام البرمجي يتألف من كتكلة واحدة

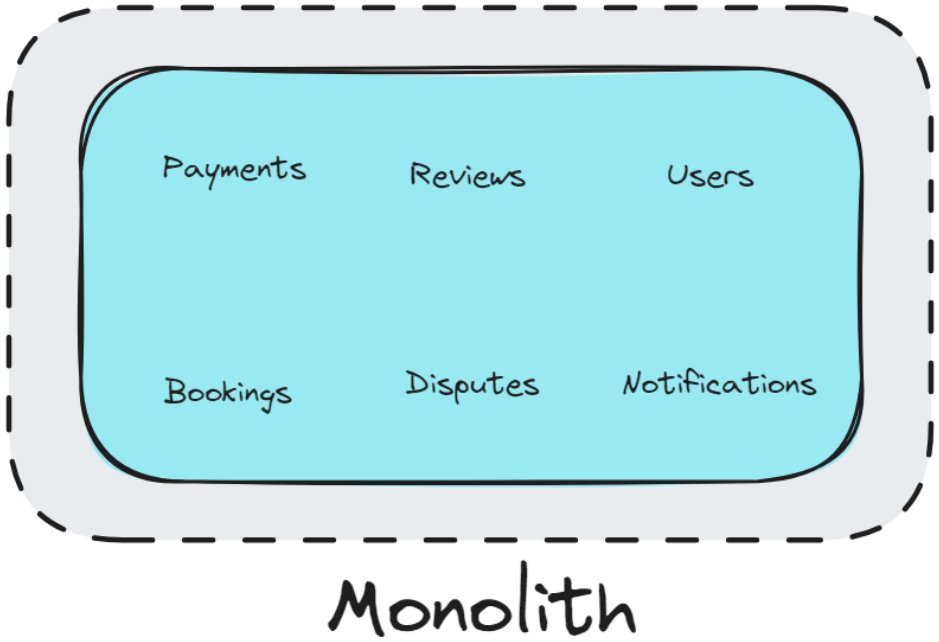
المزايا: يتميز المشروع وفق هذه البنية بالبساطة نسبيا وانخفاض التكلفة وسرعة الاداء

إلا أن عيوبها تتمثل في افتقارها للمرونة والاستجابة للتغيرات، وتحمل الأخطاء، وقابلية التوسع.

فإذا وُجد أي مشكلة في النظام ، فان ذلك يؤدي إلى تعطل النظام بأكمله وتوقفه عن العمل.

كما انه قد تواجه هذه البنية صعوبة في التكيف مع التقنيات الجديدة والمتقدمة.

مما جعل المصممين يفكرون بتجزئة النظام البرمجي الى مكونات بدلا من استخدام البنية الأحادية التقليدية. الا أن هذا الخيار مازال جيدا للتطبيقات الصغيرة ذات التعقيد المنخفض.



مثال : لنفرض لدينا hospital information system و هو نظام متكامل ، لا نستطيع فهم هذا النظام و تطويره ك block كاملة و لذلك نقوم بتقسيم هذا النظام إلى نظم جزئية مما يسهل عملية فهم و تطوير النظام و نقوم بوضع مخططات لبنية هذا النظام .

كما ذكرنا في وقت سابق معظم التطبيقات و النظم تنتمي لمجال أوسع منها يحوي تطبيقات و نظم متشابهة مع بعضها البعض و لذلك يكون لتلك النظم المتشابهة التصميم المعماري الأساسي ذاته و لذلك يقوم المصمم بخبرته بنسب هذا النظام إلى مجالة و اشتقاق المعمارية المناسبة من مجاله و الاستفادة منها و التعديل عليها .

إن خرج مرحلة التصميم هو عبارة عن وثيقة تحوي مخطط عن بنية النظام و تقسيمه إلى أنظمة جزئية (collection of sub systems) .

التجزئة Modularization: تقسيم النظام البرمجي إلى وحدات مستقلة قابلة للتبديل ويمكن تصميمها وتطويرها وإدارتها بشكل منفصل بينما تستمر في العمل معا كوحدة متكاملة لتقديم الخدمات المطلوبة من النظام.

Modularization is the process of dividing a software application into independent, interchangeable modules that can be designed, developed, and managed separately while still functioning as an integrated whole.

There are many advantages of Modularization in software engineering.

هناك العديد من المزايا والفوائد للتجزئة في هندسة البرمجيات.

- Easy to understand the system.
- System maintenance is easy.
- A module can be used many times as their requirements. No need to write it again and again.

A module is a **piece of code** that can be **independently** created and maintained to be **(re)used in different systems**.

ملاحظة هامة :

ال sub system هو system بحد ذاته و يمكنه إن يعمل بشكل مستقل و ينظر له البعض ك Modules

للتفريق بينهما سنأخذ المثال :

لنفرض لدينا hospital information system مكون من عدة sub system مثل :

1) Sub system يهتم بأمور المرضى .
2) Sub system يهتم بأمور الممرضين و الأطباء .
3) Sub system يهتم بأمور الأدوية .

و يطلق البعض أحياناً على ال sub system كلمة component و هي كلمة مرادفة ل sub system .

و ال component على نوعين :

1- Large grain .
2- Fine grain .

إن ال **large grain** هو component يقدم وظائف و مهام كبيرة أي بمعنى آخر هو code يقدم خدمات متعددة .

أما ال **fine grain** هو code يقدم مهمات صغيرة مهمة أو اثنين على الأكثر.

المكونات large-grained أكبر حجمًا وتوفر وظائف أوسع، وغالبا ما تجمع عمليات متعددة ذات صلة في خدمة أو واجهة واحدة، مثل خدمة واحدة تتولى تسجيل المستخدم وتسجيل الدخول وتحديثات الملف الشخصي. يبسط هذا النهج التصميم ويقلل من النفقات العامة، ولكنه قد يحد من المرونة والتزامن. في المقابل، تكون المكونات fine-grained أصغر حجمًا وأكثر تخصصا، وتوفر تحكما دقيقا من خلال عمليات متعددة وضيقة النطاق، مثل نقاط نهاية منفصلة لإنشاء مستخدم أو تحديث بريد إلكتروني.

في حين أن التصميمات fine-grained تعزز المرونة ، إلا أنها تزيد من التعقيد والنفقات العامة للإدارة والحاجة إلى معالجة أخطاء متطورة.

ينطوي الاختيار بين النهجين على مقايضات بين الأداء وقابلية التوسع وقابلية الصيانة والتحكم، وغالبًا ما تجمع الأنظمة المثلى بين الاستراتيجيتين بناءً على متطلبات محددة.

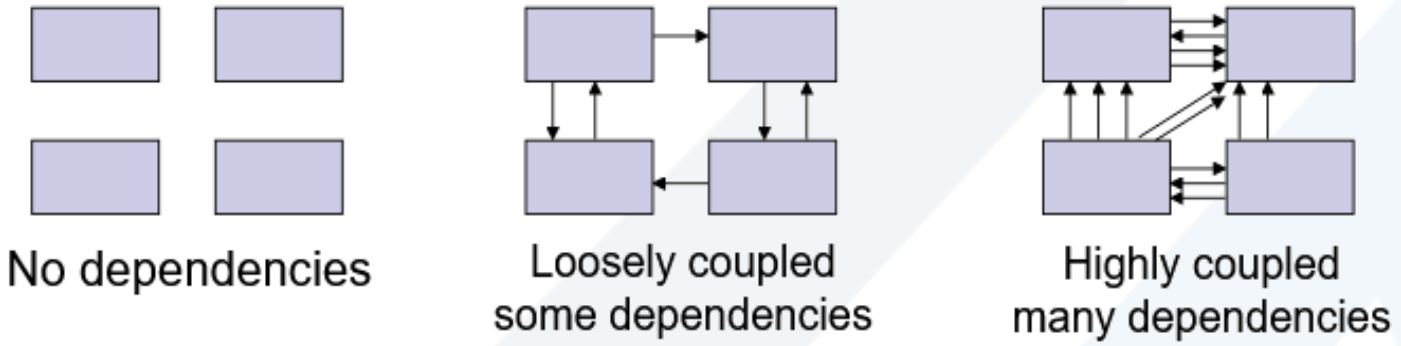
cohesion and coupling

لكن ماذا يعني كلاً من ال cohesion و ال coupling ؟

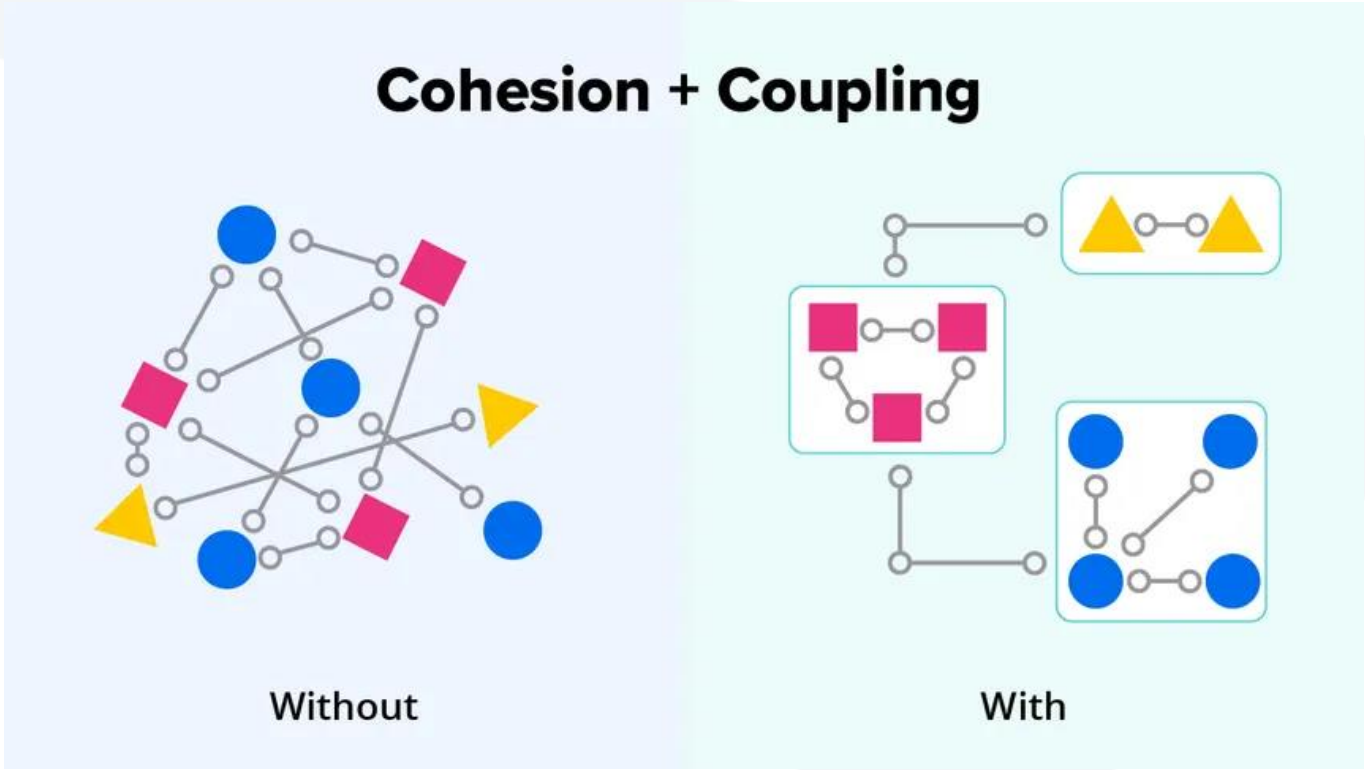
كما ذكرنا سابقاً أن ال component هو كلمة رديفة ل sub system و ذكرنا أن ال sub system يقدم خدمة أو أكثر و لذلك فإن ال component هو عبارة عن block of code يقدم خدمة واحدة أو عدة خدمات فعندما يحوي ال component جميع الخدمات و الوظائف المتعلقة ببعضها البعض أي الوظائف المتكاملة فعند طلب خدمة معينة من ال component يمكنه تلبيتها فوراً كون الخدمة و الخدمات المرتبطة بها موجودة بنفس ال component و هو ليس بحاجة إلى الاتصال مع باقي ال component في النظام لتلبية هذه الخدمة و هذا ما نعنيه بال cohesion ، أي أن الوظائف التي يقدمها ال component متلاحمة مع بعضها البعض و أكواد هذه الخدمات معتمدة على بعضها البعض و لا تعتمد على أكواد خارج ال component .

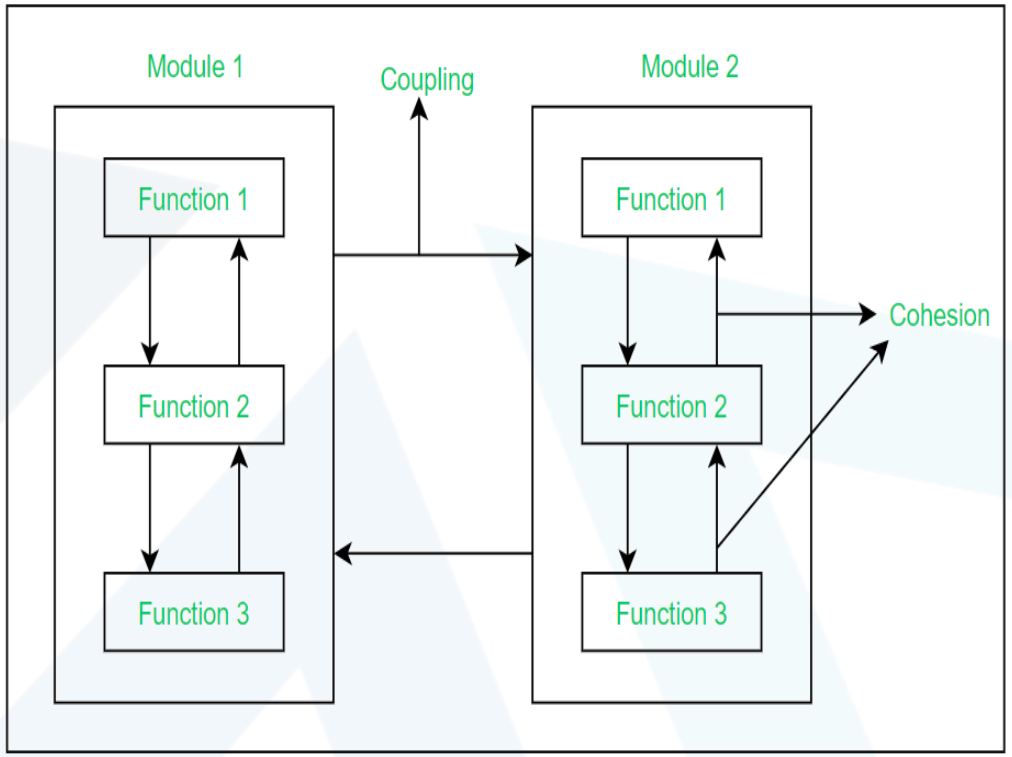
Coupling

- The degree of dependence such as the amount of interactions among components



Coupling is a measure of strength in relationship between various modules within a software





A good software design requires high cohesion and low coupling.

والتصميم الجيد والمرن يكون قادرا على التعامل مع التغيرات التي قد تحدث على مر الزمن في تكنولوجيا الأجهزة والبرمجيات وكذلك في سيناريوهات المستخدم ومتطلباته

	Cohesion	Coupling
1	Cohesion is the degree to which the elements inside a module belong together.	Coupling is the degree of interdependence between the modules.
2	A module with high cohesion contains elements that are tightly related to each other and united in their purpose.	Two modules have high coupling (or tight coupling) if they are closely connected and dependent on each other.
3	A module is said to have low cohesion if it contains unrelated elements.	Modules with low coupling among them work mostly independently of each other.
4	Highly cohesive modules reflect higher quality of software design	Loose coupling reflects the higher quality of software design

The particular architectural style should depend on the **non-functional system requirements**:

- **Performance**: Use large grain components to minimize communications.
- **Maintainability**: use fine-grain, replaceable components.

software architecture patterns

Patterns are a means of representing, sharing and reusing knowledge. An architectural pattern is a stylized description of a good design practice, which has been tried and tested in different environments.

Software architecture patterns serve as templates that guide the structuring of software systems, allowing for both efficient and manageable solutions.

What are software architecture patterns in Software Development?

1. Layered Architecture
2. Client-Server Architecture
3. Repository architecture
4. Pipe and filter architecture
5. MVC Architecture
6. Message Broker Architectural

……

جَامعة
المَنارة
MANARA UNIVERSITY