جامعة حمص
الكلية التطبيقية
قسم تقنيات حاسب
السنة الثالثة

هندسة برمجيات
SOFTWARE ENGINEERING
(SE)

د.لودا رشيد علي

فصل ثاني 2025-2026 .

هدف المقرر:

- تزويد الطالب بمفاهيم الإجرائية البرمجية وأنواعها.
- التعزيز على إدارة المتطلبات واستخدام الأدوات المناسبة لها.

Lecture1

هندسة البرمجيات

SOFTWARE ENGINEERING (SE)

مفاهيم عامة

1- مقدمة:

حتى نتعرف على هندسة البرمجيات سنفكك المصطلح إلى جزأيه: الهندسة &البرمجية

1-1- **الهندسة Engineering:** مجموعة طرائق ونظريات وأدوات ناتجة عن تجارب متكررة على مر سنوات وتعني القدرة على استثمار المعلومات في المكان المناسب لحل المشاكل بأفضل طريقة **(أقل كلفة ، أقصر زمن، أعلى جودة).**

تعتبر عوامل الكلفة والزمن والجودة معياراً أساسياً لكنه معيار متعاكس لأن أي عاملين في المعيار سيتعارضان مع العامل الثالث.

مثال: الزمن الأقصر + الكلفة الأقل يتعارض مع الجودة الأعلى.

يكون الشخص مهندساً عندما يمتلك قدرة لاتقاس بناء على المعرفة وحدها وإنما على الخبرة التي يمتلكها، فكيفية استخدام المعلومات بالشكل المناسب هو ما يجعله مهندساً قادراً على الوصول إلى حل أمثل لمشكلة ما.

1-2- **البرمجية Software**: مجموعة من الأنظمة البرمجية المستقلة (أي لكل برنامج كيانه الخاص) والمترابطة مع بعضها البعض (أي أن بينها قنوات ربط) والموثقة جاهزة للاستخدام باعتمادية عالية.

تعريف آخر:" البرمجية هي مجموعة من البرامج (العامة أو الخاصة) والوثائق المرفقة بها".

نتعلم هندسة البرمجيات لكي نمتلك الفكر الهندسي لبناء نظام معلومات بعيداً عن الناحية البرمجية وهذا ما يجعل الأنظمة البرمجية تختلف عن بعضها تبعاً لتفكير المبرمج.

2- سرد تاريخي:

يُمكن اعتبار تاريخ هندسة البرمجيات قصة تحول "الكتابة البرمجية" من مجرد فن فردي أو هواية تقنية إلى انضباط هندسي صارم تقوده المعايير.

فيما يلي سرد لأهم المحطات التي شكلت هذا المجال:

2-1- مرحلة ما قبل التأسيس (الأربعينيات – الستينيات):

في البدايات، لم يكن هناك مسمى "مهندس برمجيات". كان التركيز ينصب على الأجهزة (Hardware)، بينما كانت البرمجيات تُعتبر أمراً ثانوياً يُكتب بلغة الآلة أو التجميع (Assembly).

* برمجة مخصصة: كانت البرامج تُكتب لجهاز واحد محدد، وإذا تغير الجهاز، يجب إعادة كتابة البرنامج بالكامل.

* ظهور اللغات العالية: شهدت هذه الفترة ولادة لغات مثل Fortran و COBOL، مما جعل البرمجيات أكثر قابلية للقراءة.

2-2- أزمة البرمجيات وولادة المسمى (1968):

مع زيادة تعقيد الأنظمة في الستينيات، بدأت تظهر مشاكل ضخمة: المشاريع تتجاوز الميزانية، وتتأخر عن الموعد، والأنظمة مليئة بالأخطاء؛ عُرف هذا بـ "أزمة البرمجيات" (Software Crisis).

أزمة البرمجيات (Software Crisis) لم تكن مجرد مشكلة عابرة، بل كانت "نقطة التحول" التي جعلت العالم يدرك أن كتابة الكود ليست مجرد هواية، بل هي مسؤولية هندسية خطيرة.

2-2-1- جذور المشكلة (لماذا حدثت؟)

في الستينيات، تطورت قدرات الأجهزة (Hardware) بسرعة هائلة (ظهور الدوائر المتكاملة)، مما سمح ببناء حواسيب أقوى؛ لكن القدرات البشرية في كتابة البرامج (Software) لم تتطور بنفس السرعة.

* التعقيد المتزايد: طُلِب من المبرمجين كتابة أنظمة ضخمة (مثل أنظمة التشغيل أو أنظمة التحكم في الطيران) باستخدام أدوات بدائية.

* غياب المنهجية: لم تكن هناك قواعد واضحة لكيفية إدارة مشروع برمجِي؛ كان المبرمج يبدأ بالكتابة فوراً دون تخطيط.

2-2-2- أعراض الأزمة (ماذا حدث فعلياً؟)

في ذلك الوقت، كانت معظم مشاريع البرمجيات تعاني من أربع كوارث:

* تجاوز الميزانية: كانت التكاليف النهائية تضاعف الميزانية المرصودة عدة مرات.

* التأخير الزمني: المشاريع التي كان من المفترض أن تستغرق سنة، كانت تستغرق ثلاث سنوات أو لا تنتهي أبداً.

* الجودة السيئة: البرامج كانت مليئة بالأخطاء (Bugs) وغير مستقرة.

* صعوبة الصيانة: بمجرد مغادرة المبرمج الأصلي، يصبح الكود "طلاسم" لا يفهمها أحد، مما يجعل تعديله مستحيلاً.

2-2-3- المثال الأشهر: نظام OS/360: أكبر مثال على هذه الأزمة كان نظام التشغيل OS/360 من شركة IBM.

* استثمرت فيه الشركة آلاف المبرمجين وملايين الدولارات.

* النتيجة كانت تأخراً هائلاً وكوداً معقداً جداً.

* خرج منه "فريد بروكس" بكتابه الشهير "The Mythical Man-Month"، حيث صاغ قانونه الشهير: "إضافة مبرمجين لمشروع برمجِي متأخر، سيزيده تأخراً."

2-2-4- الحل: ولادة "هندسة البرمجيات"

في عام 1968، اجتمع قادة التقنية في مؤتمر الناتو في مدينة غارمش بألمانيا وقرروا أن الحل الوحيد هو التوقف عن معاملة البرمجة كـ "فن" أو "سحر"، وبدء معاملتها كـ "هندسة".

ماذا نتج عن ذلك؟

* استعارة مفاهيم من الهندسة المدنية (مثل التخطيط، التصميم قبل التنفيذ، التوثيق).

* ابتكار نماذج دورة حياة البرمجيات (SDLC).

* التركيز على قابلية الاختبار وقابلية الصيانة، وليس فقط أن البرنامج "يعمل".

وهكذا استُخدم مصطلح "Software Engineering" لأول مرة بشكل رسمي؛ كان الهدف هو التأكيد على أن تطوير البرمجيات يحتاج إلى نفس الانضباط الذي تحتاجه الهندسة المدنية أو الميكانيكية

2-3- عصر النماذج الهيكلية (السبعينيات - الثمانينيات):

في محاولة للسيطرة على الفوضى، بدأ الخبراء في ابتكار نماذج لإدارة دورة حياة النظام.

* نموذج الشلال (Waterfall): قدمه "وينستون رويس" (رغم أنه انتقد صرامته لاحقاً)، اعتمد هذا النموذج على خطوات خطية (تحليل، تصميم، تنفيذ، اختبار).

* البرمجة المهيكلة: برزت دعوات لترك استخدام GOTO والاعتماد على هياكل تحكم واضحة (IF, Loops) بقيادة علماء مثل "إدزجر ديكسترا".

2-4- ثورة الكائنات والنمذجة (التسعينيات):

مع تطور أجهزة الكمبيوتر الشخصية، أصبحت البرامج أضخم وأكثر تعقيداً، مما أدى لظهور مفاهيم جديدة:

* البرمجة الكائنية (OOP): أصبحت لغات مثل C++ ولاحقاً Java هي المعيار، حيث يتم تنظيم الكود حول "البيانات" أو "الكائنات" بدلاً من "الوظائف".

* لغة النمذجة الموحدة (UML): ظهرت لتوحيد الطريقة التي يرسم بها المهندسون مخططات الأنظمة قبل بنائها.

2-5- عصر "Agile" والسرعة (2001 - الآن)

شعر المطورون أن النماذج القديمة (كالتحليل المطول قبل التنفيذ) لم تعد تناسب سرعة عصر الإنترنت.

* بيان الأجايل (Agile Manifesto): في عام 2001، اجتمع 17 مطوراً وأعلنوا مبادئ "الأجايل"، التي تركز على المرونة، التسليم السريع، والتفاعل مع العميل بدلاً من التوثيق المكثف.

* العمليات المستمرة (DevOps): في العقد الأخير، تلاشت الحدود بين "التطوير" و"العمليات"، وظهرت مفاهيم مثل التكامل المستمر (CI) والنشر المستمر (CD).

فيما يلي ملخص التطور التاريخي

الحقبة	التركيز الأساسي	المنهجية البارزة
1950 – 1960	الأجهزة (Hardware)	البرمجة الخطية البسيطة
1970 – 1980	السيطرة على التعقيد	نموذج الشلال والبرمجة المهيكلة
1990 – 2000	إعادة الاستخدام والنمذجة	البرمجة الكائنية (OOP) و UML
2001 – الآن	السرعة والتكيف	الأجايل (Agile) و DevOps

نلاحظ تحول هذا المجال من "فن كتابة الأكواد" إلى "علم إدارة الأنظمة المعقدة".

3- البرامج الحاسوبية Computer Program:

برامج معدة من المبرمج لكنها ليست نهائية أي أنها برامج أولية يكون الهدف فيها المبرمج (لا يستطيع التعامل معها إلا المنتج لها)، أي أن **Computers Programs** أو البرامجيات تكون مرحلية ومن ثم تتحول إلى **Software** فور الانتهاء منها بالصيغ التالية:

- **الإيعازات(التعليمات) Instructions:** عندما يتم تنفيذها تعطي الوظيفة المطلوبة.
- **بنى المعطيات Data Structures:** لتمكين البرامج من معالجة البيانات بشكل كافٍ.
- **الوثائق Documents:** توصف تشغيل واستخدام البرنامج.

4- هندسة البرمجيات Software Engineering (SE) :

عملية بناء واستخدام المبادئ الهندسية المعروفة من أجل الحصول على برنامجٍ اقتصاديّ، يتميز بكونه موثوق ويعمل بكفاءة على حواسيب حقيقية.

وضع معهد مهندسي الكهرباء والإلكترونيات (IEEE) (Institute of Electrical & Electronics Engineers) تعريفاً أشمل:

"هي الفرع الهندسي المختص باستخدام الطرق المنهجية والقابلة للقياس، من أجل تطوير وتشغيل وصيانة البرامج. أي أنها علم تطبيق الهندسة في مجال البرامج."

تهتم هندسة البرمجيات عموماً بتحليل البرمجيات وتصميمها وبنائها وتحقيقها وإدارتها، وللقيام بهندسة برمجيات صحيحة يجب تعريف الإجرائية البرمجية **software process**، أي المنهج المتبع لهندسة البرمجيات.

بناء النظام البرمجي في هندسة البرمجيات ليس مجرد كتابة شيفرة (كود Code)، وإنما هي عملية إنتاجية لها عدة مراحل أساسية وضرورية للحصول على المُنتج.

الهدف النهائي من أي برنامج هو الحصول على أفضل كفاءة ممكنة بأقل كلفة ممكنة.

تجسدت العوامل التي ساعدت على تبني مفهوم هندسة البرمجيات بما يلي:

- ارتفاع تكاليف صناعة البرمجيات مقارنة مع المكونات المادية للحاسوب.

- الطلب المتزايد على البرمجيات بما في ذلك الكبيرة والمعقدة والدور المتزايد لصيانة تلك البرمجيات.
- التقدم السريع في التقانات الحاسوبية.
- التقدم في أساليب بناء البرمجيات.

5- الإجرائية البرمجية Software Process:

مجموعة أنشطة مرتبة هدفها النهائي تطوير منتج برمجي جديد أو تحسين منتج موجود.

تتمحور كل الإجرائيات البرمجية حول أنشطة عامة رئيسية:

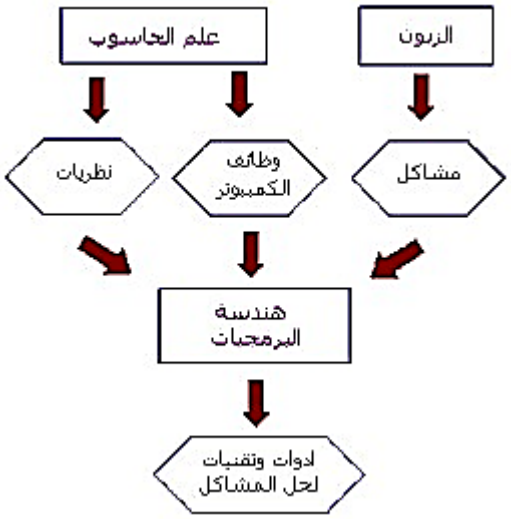
التحليل–التصميم–التحقيق

أما المراحل التفصيلية فهي:

تحديد المتطلبات– توصيف المتطلبات– تصميم البنيان– التصميم التفصيلي– المكاملة– الصيانة.

6- هندسة الأنظمة والبرمجة Programming:

هندسة الأنظمة تشمل جميع جوانب تطوير الأنظمة الحاسوبية ومنها Hardware، البرمجيات، العمليات، الأشخاص، وبالتالي فان هندسة البرمجيات هي جزء من هندسة الأنظمة.



الشكل(1-1) العلاقة بين هندسة البرمجيات وعلم الحاسب

تختلف هندسة البرمجيات عن **البرمجة**، المبرمج وحده يكتب برنامجاً كاملاً بينما مهندس البرمجيات يعتني بجزء من نظام متكامل لينضم هذا الجزء إلى أجزاء يعتني بها مهندسو برمجيات آخرون، كما يمكن تعديل جزء مكتوب من طرف غير الذي كتبه.

البرمجة يمكن أن تكون عملاً فردياً بينما هندسة البرمجيات عمل فريق، البرمجة لا يمكن التنبؤ بموعد انتهاء البرنامج بينما الهندسة فتقدم جداول زمنية، تقديرات مالية، وتحليلات للمخاطر.

7- مهندس البرمجيات Software Engineer:

الشخص الذي تقع على عاتقه مهام تطوير المنتجات البرمجية بأسلوب منظم يحقق الأهداف المرجوة باستخدام الأدوات المناسبة، كاختيار لغة برمجة مناسبة من لغات البرمجة عالية المستوى، والتقنيات الضرورية كالخوارزميات متعددة الأغراض حسب نوع المشكلة التي يقوم بحلها وقيود التطوير المفروضة والموارد المتاحة

(المادية والبشرية)، لا يهتم مهندس البرمجيات فقط بالعمليات التقنية لتطوير البرمجيات بل يهتم أيضاً بالأنشطة مثل إدارة مشروع البرمجيات وتطوير الأدوات والطرق والنظريات التي تدعم إنتاج البرمجيات.

8- **الطبقات التقنية لهندسة البرمجيات:** الهدف الرئيس من هندسة البرمجيات هو إنتاج أنظمة برمجيات ذات جودة عالية ويمكن توضيح الطبقات التي تعمل وفقها بالشكل التالي:

- **طبقة الأدوات Tools Layer** تساعد في تسريع العمل وتحسين الدقة ؛ **تنقسم** إلى:
  - **Software** اللغة البرمجية التي سوف نستخدمها في إنشاء نظام أكثر اعتمادية.
  - **Hardware** الأدوات التي سوف تستخدم للنظام من قطع ولوازم وتجهيزات.

تشمل البرامج التي تدعم تطوير البرمجيات (بيئات التطوير (IDE)، أدوات إدارة الإصدارات مثل Git، أدوات الاختبار).

- **طبقة الطرائق Methods** هي الطرق أو الخوارزميات المختلفة لإنشاء وتصميم النظام أو البرنامج وكيفية تدفق البيانات؛ تشمل تحليل المتطلبات، التصميم، كتابة الكود، الاختبار.

تركز على "كيف" يتم بناء النظام.

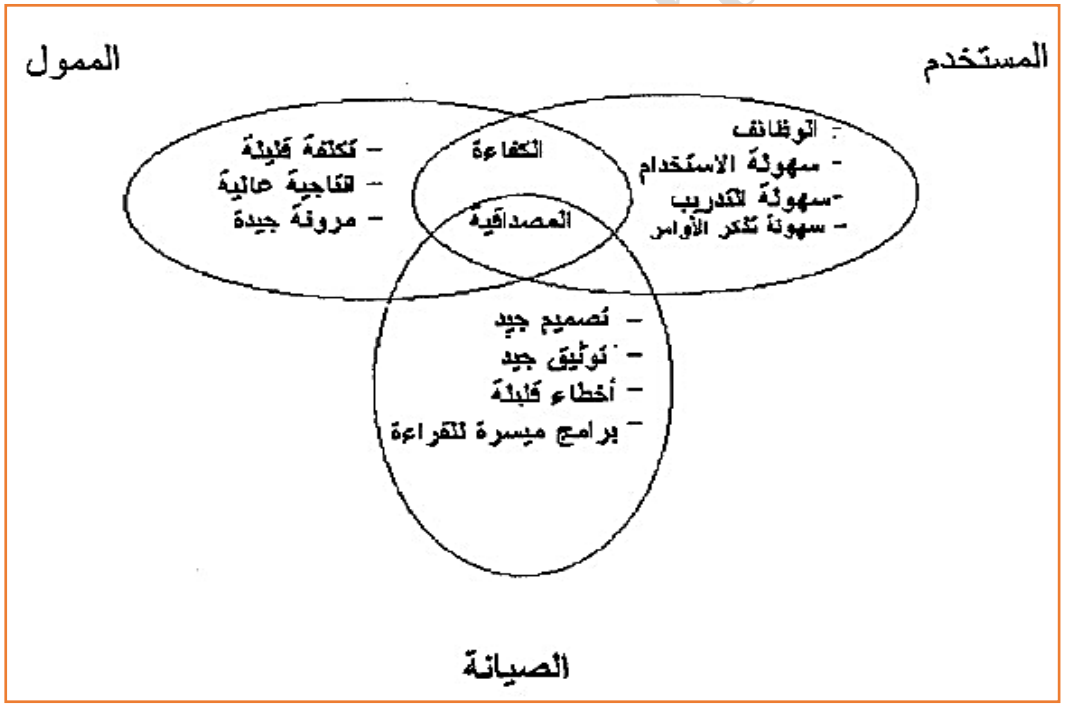
الشكل(1-2) طبقات هندسة البرمجيات

- **الطبقة الثالثة إجرائيات التطوير Development Process أو طبقة العمليات Process Layer**
تعبر عن الكود البرمجي الذي سيقوم بتطوير النظام الحالي يدوياً أو آلياً.
نمثل الإطار الذي ينظم خطوات تطوير البرمجيات، تشمل نماذج مثل الشلال Waterfall، الرشيقة Agile.
تساعد الفرق على العمل بشكل منظم وتقليل الأخطاء.
- **الطبقة الأخيرة Quality Focus** تعتبر الأساس في كل عملية تطوير، تهدف إلى ضمان أن البرنامج يلبي المتطلبات ويعمل بكفاءة، تشمل معايير الاختبار والتحقق وتؤكد أنه يجب التركيز على الجودة في كل مما سبق بحيث أن كل طبقة من الطبقات السابقة مرتبطة ومعتمدة على الأخرى.

عندما تبدأ مرحلة تشغيل نظام البرمجيات، فإن المفاهيم تتداخل بين ثلاث جهات مشاركة في عملية صناعة البرنامج هم "أصحاب المصلحة" **(Stakeholders)** :

- الممول (المالك) **Customer** الذي يتحمل تكاليف تطوير البرمجيات.
- المستخدم **User** لهذه البرمجيات.
- المطور (أو فريق الصيانة) **Developer** يقوم بالتعديلات وتصحيح الأخطاء.

يختلف مفهوم الجودة بين الشركاء الثلاثة حيث يهدفون معاً للمصداقية، المستخدم والممول يهتمان معاً بالكفاءة أما على ماذا يركز كل طرف فيظهر في الشكل التالي:



الشكل(1-3) الجودة من وجهات نظر الشركاء في صناعة البرنامج

9- الخصائص البارزة للنظام:

نعلم أن النظام (System) هو مجموعة من العناصر والقواعد المترابطة (الوحدات) التي تعمل معا لأداء مهمة (وظيفة) معينة، **النظام الجزئي (الفرعي) Sub-System** هو نظام قائم بذاته يعتمد عليه نظام آخر ويكون جزء من نظام متكامل.

عموماً تتضح الخصائص البارزة للنظم في:

- البرامج
- ملفات التكوين (تسمى أحياناً البيانات) التي بتم استخدامها لإعداد البرامج.
- مستندات توثيق النظام التي تصف هيكل النظام وتحتوي على وثائق تعليمات استخدام النظام الموجهة للمستخدمين.

أهم الأمور التي يجب على المبرمجين أخذها بعين الاعتبار عند القيام ببرمجة نظام معين هي **خصائص النظام – الكفاءة التي يعمل بها – الأداء** بالمعايير التالية:

- الاستخدام Usability: فهل يسمح به على نطاق عريض أم مخصص.
- كفاءة النظام **Efficiency**: الإنجاز أو أداء النظام لقيامه بعمليات محددة تحت حمل أو ضغط معين.
- الاعتمادية **Performance**: أي هل يمكن الاعتماد على هذا النظام أم لا
- الواقعية **Reliability** : قابلية النظام للتحقيق واقعياً
- محمول **Portability**: قابلية النظام للتشغيل و التحميل على عدة أجهزة و أنظمة مختلفة
- **Traceability** عملية الاحتفاظ بنسخة أصلية عن كل عملية تعديل

10- مكونات النظام:

أول خطوات تحليل المشكلة هو فهم ماهيتها وتعريفها بوضوح، لذا يجب أولا وصف النظام بتحديد مكوناته والعلاقات التي تربط بين هذه المكونات.

النشاط هو عملية تحدث في النظام، وعادة ما يوصف كحدث يتم من خلال حافز، النشاط يغير شيئا ما إلى آخر بتغيير خواصه (صفاته)، هذا التغير يمكن أن يعني تحويل أحد عناصر البيانات من موقع إلى آخر أو تعديل قيمته إلى قيمة مختلفة، العناصر (**الكينونات أو الكائنات)** عادة تكون مرتبطة ببعضها البعض بشكل أو بآخر مثالها: الكائنات المرتبة في مصفوفة-أو سجل.

يجب وضع **وصف الكائنات** بدقة (نوعها، النشاطات التي يمكن إجراؤها عليها).

تعريف الكائن يتضمن:

- الموقع الذي سوف ينشأ منه (بعض العناصر يمكن أن تكون موجودة بملف سبق إنشاؤه، والبعض قد يتم إنشاؤه خلال حدث ما).

- الهدف من إنشائه (بعض الكائنات تستخدم من نظم أخرى كمدخلات).

العلاقات وحدود النظام بعد تعريف الكائنات والنشاط جيداً يمكن أن نربط بين كل كائن والنشاطات المتعلقة به بدقة حيث يوجد لأي نظام **حدود** بعض الكائنات، يمكن أن تعبر هذه الحدود إلى داخل النظام والبعض الآخر هي مخرجات من ذلك النظام يمكن أن ترحل إلى نظم أخرى.

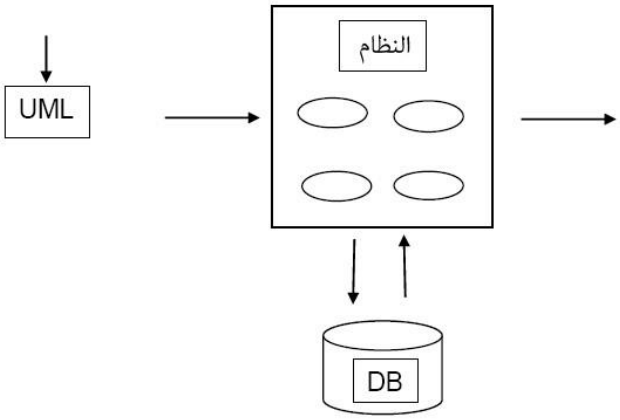
بهذا يمكن أن نعرف النظام بأنه تجمع من:

- **Entities** كائنات (كينونات)
- **Activities** أنشطة
- **Relationships** وصف للعلاقات بين الكائنات والأنشطة
- **Boundary** تعريف لحدود النظام

بمعنى: أي نظام هو عبارة عن مجموعة من الكائنات (الكينونات) Objects والنشاطات Activities بالإضافة إلى وصف للعلاقات التي تربط تلك الكائنات والنشاطات معا مع تعريف قائمة المدخلات المطلوبة والخطوات المتبعة والمخرجات الناتجة لكل نشاط.

يجب التمييز بين قاعدة المعطيات ونظام المعلومات حيث أن قاعدة المعطيات تمثل ذاكرة نظام المعلومات وهي ليست بديلة عنه.

من أمثلة نظم المعلومات: نظام الحكومة الإلكترونية– نظام حجز طيران– نظام إصدار برنامج الدوام في الكلية الذي دخله (المواد – القاعات –المدرجات) وخرجه البرنامج، أما مهامه فهي إمكانية التعديل كإرسال رسائل للدكاترة...



الشكل(1-4) النظام وقاعدة البيانات