لغة النمذجة الموحدة
(UML)

مقدمة

- تقدم لغة النمذجة الموحدة مجموعة من أفضل خبرات الممارسات الهندسية التى ثبت نجاحها فى نمذجة النظم الكبيرة والمعقدة، وهى جزء مهم من التطوير الكائنى المنحى للبرمجيات وعمليات تطوير البرمجيات.
- ليست لغة النمذجة الموحدة منهجية لبناء أو تصميم البرمجيات وتطويرها.
- لا ترتبط لغة النمذجة الموحدة بمنهجية أو طرق إنتاج البرمجيات، ويمكن توظيف هذه اللغة على مختلف العمليات البرمجية بغض النظر عن المنهجية المتبعة.
- تتألف لغة النمذجة الموحة من أربع طبقات أساسية تقسم كل طبقة بدورها إلى طبقات فرعية

• لغة (UML) :

هي اختصار إلى (Unified Modeling Language) .
باللغة العربية تسمى (لغة النمذجة الموحده).

• استخدام لغة (UML) :

في عمل التخطيط لمشاريع البرمجة قبل البدا في عمل المشروع البرمجي عملياً.

• مكونات لغة (UML) :

١. أشكال هندسية .
٢. رموز مثل (مربع ، مستطيل ، خط مستقيم ، أسهم ، سداسي) .

ملاحظة :- كل رمز من الرموز يرمز إلى شيء في المشروع البرمجي.

• ابرز سمات لغة (UML) :

١. لا ترتبط لغة النمذجة الموحدة بمنهجية أو طرق إنتاج البرمجيات .

٢. تقدم لغة النمذجة الموحدة مجموعة من أفضل خبرات الممارسات الهندسية .

٣. ليست لغة النمذجة الموحدة منهجية لبناء أو تصميم البرمجيات وتطويرها.

لماذا تستخدم لغة النمذجة الموحدة UML

و بناءا على استراتيجية زيادة اهمية البرامج للعديد من الشركات ، فمجال الصناعة سعى بالبحث عن تقنيات لجعل إنتاج البرمجيات أوتوماتيكي، مع تحسين النوعية والضغط على التكلفة والوقت لزيدة القدرة التنافسية في السوق. وتشمل هذه التقنيات تقنية المركبات " component technology"" والبرمجة المرئية "visual programming"، و القوالب "patterns" ومنصات العمل " frameworks". الشركات تسعى أيضا إلى تقنيات لإدارة تعقيدات الأنظمة لأنها في زيادة من حيث الحجم و المدى. وعلى وجه الخصوص ، فهي تعترف بالحاجة إلى حل المشاكل الهندسية المتكررة ، مثل التوزيع المادي " physical distribution" ، التزامن "concurrency" التكرار "replication"، الأمن ، الموازنة "load balancing" و بالإضافة إلى ذلك ، قد أدى تطور شبكة الويب العالمية ، "fault tolerance" الاحتمال الخطأ " مما جعل بعض الأمور أكثر بساطة ، في زيادت المشاكل الهندسية. وقد تم تصميم لغة النمذجة الموحدة "UML" للاستجابة لهذه الاحتياجات.

من أربع طبقات أساسية UML تتألف لغة النمذجة

١. طبقة كائنات المستخدم User Objects Layer

هي الطبقة السطحية العامة التي يستخدمها الذين يتعاملون مع لغة النمذجة الموحدة، وتتألف من تسعة مخططات رئيسية بالإضافة إلى كائنات وأدوات مساعدة
وهى الطبقة الأكثر وضوحاً وتوصيفاً، المستخدم هنا المقصود هو مستخدم اللغة وليس المستخدم النهائي للبرمجية أو المنتج البرمجي .

تشمل الطبقة الأولى المخططات التسعة التالية :

* مخطط حالات لاستخدام.
* مخطط الفئة.
* مخطط الكائن.
* مخطط النشاط.
* مخطط التعاون.
* مخطط الحالة.
* مخطط التتابع .
* مخطط المكونات .
* مخطط التوزيع.

٢. الطبقة الثانية :تسمى طبقة النموذج Model Layer

وتكون فى المرحل الأولى من التحليل حيث تحتوى على مفاهيم موضوع التحليل كفهم النظام ويستخدم هذه الطبقة محلل النظام أثناء عمله قبل نضج فكرة النظام أو وضوحه بتوصيف أقل من طبقة المستخدم الأولى.

٣. الطبقة الثالثة: طبقة ما وراء النموذج Meta Model Layer

وتعنى هذه الطبقة بالمفاهيم المتعلقة بلغة النمذجة الموحدة كمفهوم الصنف والظاهرة ونوع البيانات والتجريد والأنماط وغيرها من مفاهيم اللغة، وهى طبقة تصف ما يجرى فى النموذج وتتألف من ثلاث حزم رئيسية هى:

* حزمة الأساس .
* حزمة العناصر السلوكية.
* حزمة إدارة النموذج

٤. الطبقة الرابعة : طبقة ما وراء ما وراء النموذج (Meta meta model layer)

طبقة لا تهم معظم محللي النظم تهتم بلغة كتابة النمذجة الموحدة، وتهم مطوري أدوات لغة النمذجة الموحدة مثل البرامج التي تؤتمت المخططات وترسمها .

١. مخطط حالة الاستخدام (Use Case Diagram) :
يعرض العلاقة بين الجهات الفاعلة (actors) وحالات الاستخدام(use cases).

٢. مخطط الفئة (Class Diagram):
نماذج هيكل الفئة (class structure)ومحتوياتها تستخدم عناصر التصميم مثل
الفئات(classes)، والحزم والكائنات (objects).

ويعرض أيضا علاقات مثل الاحتواء(containment) ، التوريث(inheritance) ، و التجميع (associations) وغيرها.

٣. مخطط التفاعل (Interaction Diagrams):

- **مخطط التسلسل (Sequence Diagram)**

يعرض التسلسل الزمني للكائنات (objects) المشاركة في التفاعل (interaction).
و هذا يتألف من البعد العمودي (الوقت) والبعد الأفقي (الكائنات المختلفة — different objects) .

- **مخطط التعاون (CollaborationDiagram)**

يعرض التفاعل المنظم حول الكائنات "objects" وعلاقاتها مع بعضها البعض. تستخدم الأرقام لإظهار تسلسل الرسائل.

٤ . مخطط الحالة (State Diagram) :

يعرض تسلسل الحالات التي يمر من خلالها كائن التفاعل (object of an interaction) في حياته في عملية الاستجابة لمحفزات وردت سابقا ، جنبا إلى جنب مع ردودها والإجراءات.

٥. مخطط النشاطات (Activity Diagram):

يعرض المخططات المميزة لمخطط الحالة (state diagram) ، بحيث ان معظم الحالات هي حالات العمل (action states) ومعظم الانتقالات يتم تشغيلها بواسطة إنهاء الإجراءات في مصدر الحالات. هذا المخطط يركز على التدفقات المدفوعة من المعالجات الداخلية (internal processing).

٦. المخططات المادية (Physical Diagrams) :

- **مخطط المركب (Component Diagram)**

يعرض المستوى المرتفع لهيكل حزمة الشفرة نفسها (packagedstructure ofthe code). بالاعتماد على المركبات (components) التي ظهرت ، بما في ذلك مركبات شفرة المصدر (sourcecode components) والمكونات البرمجية الثنائية (binary code components) ، والمكونات القابلة للتنفيذ (executable components).

- **تخطيط النشر (Diagram Deployment)**

يعرض التكوين (configuration) لعناصر التجهيز وقت التشغيل (run-time processing) ومكونات البرامج (software components) ، والعمليات (processes)، والكائنات (objects) التي تعمل فيها. حالات مكونات البرامج (Software component instances) تقدم مظاهر وقت التشغيل لوحدات التعليمات البرمجية (codeunits).

المحاضرة الثانية
المنحى الكائني

د. فادي تركاوي

- في هذه المحاضرة سوف نلقي نظرة على مفهوم المنحى الكائني Object Orientation (OO) . لقد تمّ تصميم لغة النمذجة الموحّدة UML بحيث تدعم المنحى الكائني ، لذلك سنقوم بتعريف مفاهيمه قبل التعمق في لغة UML ، و استكشاف المزايا التي قد يوفرّها.
- سوف نستعمل جملة المنحى الكائني للتعبير عن المنحى الكائني للتصميم أو/ و المنحى الكائني للبرمجة.

البرمجة الهيكلة (أحيانا يُسمّى وظائفي Functional)

- في البرمجة المهيكلة Structured Programming، الطريقة العامة المتّبعة هي النظر إلى المسألة، ثم تصميم مجموعة من الوظائف functions التي يمكنها انجاز المهام المطلوبة لحلها. إذا تضخّمت هذه الوظائف ، يتم تجزئتها حتى تصير صغيرة بالحدّ الذي يتيسّر فيه مناولتها و فهمها. هذه العملية تدعى التفكيك الوظائفي functional decomposition.
- ستحتاج معظم الوظائف إلى بيانات من نوع ما لتعمل عليها. البيانات في الأنظمة الوظائفية عادة ما يحتفظ بها في قاعدة بيانات من نوع ما (أو قد يحتفظ بها في الذاكرة كمتغيّرات عامة global variables).
- لنأخذ مثالا بسيطا، تخيّل منظومة لإدارة معهد، هذه المنظومة تحتفظ ببيانات الطلبة و المدرّبين في المعهد إضافة للمعلومات حول الدورات المتوفرة، كذلك تقوم المنظومة بتتبع كل طالب و المقررات التي التحق بها.

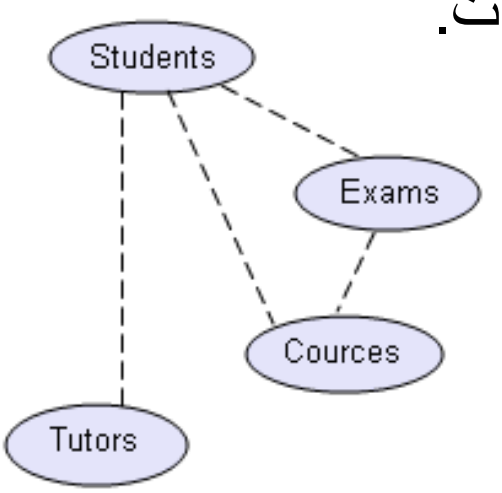
الوظائف Function

- التصميم الوظائفي المحتمل سيتضمّن كتابة الوظائف functions التالية:

add_student إضافة طالب
enter_for_examدخول امتحان
check_exam_marksفحص علامات امتحان
issue_certificateإصدار شهادة
expel_studentطرد طالب

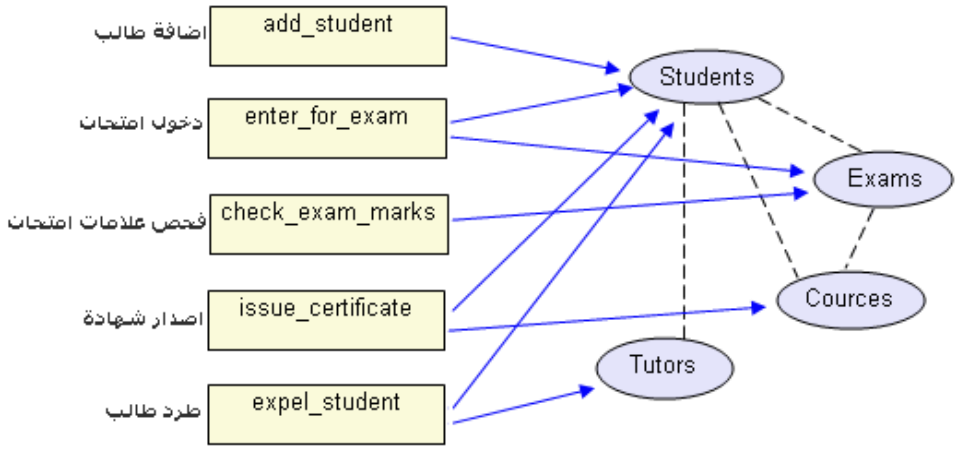
نموذج البيانات Data Model

- سوف نحتاج أيضا إلى نموذج بيانات data model ليمثّل هذه الوظائف. نحتاج لتخزين معلومات عن الطلبة، و المدربين و الامتحانات و المقررات، لذا يجب علينا تصميم مخطط قاعدة بيانات database schema للاحتفاظ بهذه البيانات.



- الآن بدأ واضحا أن الوظائف functions التي حدّدناها سابقا سوف تعتمد على هذه المصفوفة من البيانات.
مثلا، وظيفة "add_student" (إضافة طالب) ستقوم بتغيير محتويات "Students" (طلبة)، و وظيفة "issue_certificate" (إصدار شهادة) تحتاج إلى الوصول إلى بيانات طالب "Student" (لمعرفة تفاصيل الطالب الذي يحتاج للشهادة) و ستحتاج الوظيفة أيضا إلي بيانات الامتحانات "Exam" .

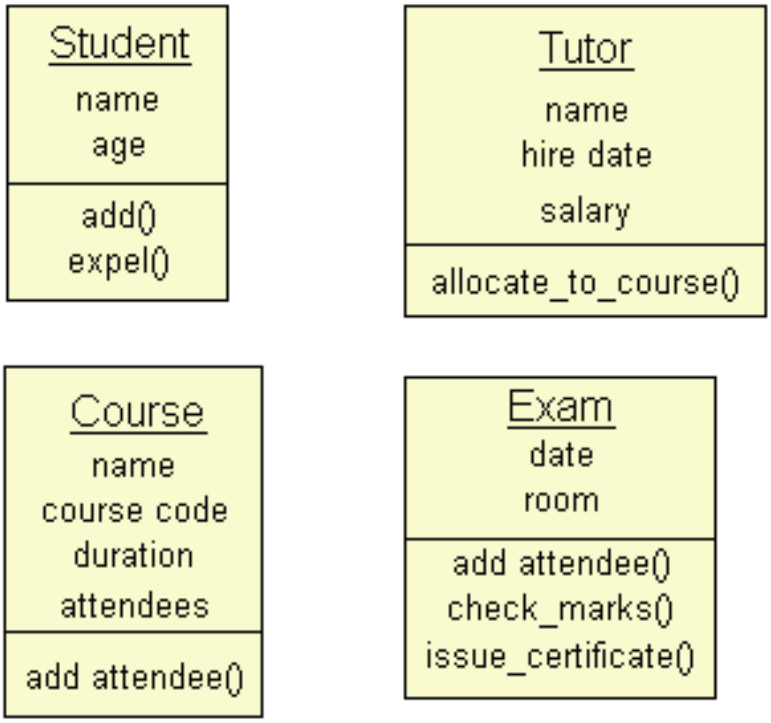
- المخطط diagram التالي عبارة عن رسم لكل الوظائف، مجتمعة رفق البيانات، و رسمت الخطوط فيها حيثما وجدت اعتمادية



- المشكلة مع هذه المقاربة؛ أن المسألة التي نتعامل معها إذا ما **تعقدت أكثر فإن صعوبة المحافظة على المنظومة** و صيانتها ستزداد. فإذا أخذنا المثال أعلاه، ماذا سيحدث لو تغيّرت المتطلبات requirements بطريقة تؤدّي إلى تغيير أسلوب مناولة بيانات الطالب Student.
- كمثال، لنتخيّل أن منظومتنا تعمل على أكمل ما يكون، لكننا اكتشفنا أن تخزين تاريخ ميلاد الطالب على شكل عدد ذو خانتين كي يمثل السنة كانت فكرة سيّئة، الحل المبدئي هنا هو أن نقوم بتغيير حقل تاريخ الميلاد في جدول الطلبة Students من خانتين إلى أربع خانات لرقم السنة.

أسلوب المنحى الكائني

- يحاول المنحى الكائني "OO" التقليل من تأثير هذه المشكلة عن طريق الجمع بين البيانات data و الوظائف functions ذات العلاقة في قالب module واحد.
- بالنظر إلى الشكل السابق ، يبدو واضحا وجود علاقة بين البيانات و الوظائف؛ فمثلا وظيفتا: add_student و expel_student مرتبطتان بقوة ببيانات Student الطالب.
- الشكل التالي يبيّن التجميعات الكلّية للبيانات و الوظائف ذات العلاقة على شكل قوالب:



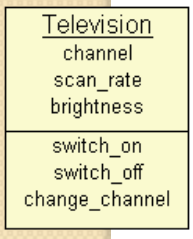
- بعض النقاط الجديرة بالملاحظة حول نظام القولبة الجديد في البرمجة:
- أكثر من تمثل لنفس القالب يمكن استحضاره عند تشغيل البرنامج. في منظومة المعهد، سيكون هناك تمثيل لقالب Student لكل طالب يتبع المعهد يجرى التعامل معه في المنظومة. و كل تمثل سيكون له بياناته الخاصة. (بالطبع لكل تمثل اسما مختلفا).
- القوالب يمكنها "التخاطب" مع قوالب أخرى عن طريق استدعاء وظائفها. مثلا، عندما يتم استدعاء وظيفية "add" (إضافة) في Student (الطالب) ، سيتم خلق تمثلا جديدا لقالب Student، و بعدها سيتم استدعاء وظيفة "add_attendee" (إضافة مشترك) من التمثل المناسب لقالب "Course" دورة.

Encapsulation التغليف

- الأمر الأساسي هنا، أنه لا يتم السماح بقراءة أو تغيير أي عنصر في البيانات إلا من قبل التمثيّل الذي تتبعه هذه البيانات . فمثلا، التمثل الخاص بقالب المدرّب Tutor لا يمكنه تحديث أو قراءة بيانات "age" العمر داخل قالب Student الطالب. هذا المفهوم يسمّى بالتغليف Encapsulation ، الذي يسهم في جعل هيكل المنظومة أكثر متانة، و يجنبها الحالة التي تعرّضنا لها سابقا، حيث التغيير البسيط في جزء من البيانات قد يؤدّي إلى تغييرات متتابعة و أكثر عمقا.
- بواسطة هذا التغليف، يمكن للمبرمج الذي يتعامل مع قالب Student مثلا أن يقوم بتغيير بيانات القالب بكل أمان و دون الخشية من وجود قوالب أخرى مرتبطة أو تعتمد على هذه البيانات. قد يحتاج المبرمج إلى تحديث الاجرائيات داخل القالب، و لكن التأثير سيبقى محصورا داخل قالب واحد معزول عن القوالب الأخرى.

الكائنات Objects

- هذه التجميعات من البيانات و الوظائف المرتبطة بأنها قوالب "modules". عموما إذا نظرنا إلى خصائص هذه القوالب سنجد مرادفات لها في العالم الحقيقي.**الكائنات** Objects في عالم الواقع يمكن تمييزها بشيئين : كل كائن في عالم الواقع لديه **بيانات** data و **سلوك** behaviour . فمثلا جهاز التلفاز هو كائن و يعالج البيانات بطريقة تجعلها تنضبط في قناة محددة، يتم تحديد معدّل المسح إلى قيمة معيّنة، كذلك معدّل التباين و شدّة الإضاءة و هكذا. التلفاز أيضا يمكنه أن "يقوم" بأشياء، التلفاز يمكنه التشغيل أو الإقفال، القنوات يمكن تغييرها، و هكذا.

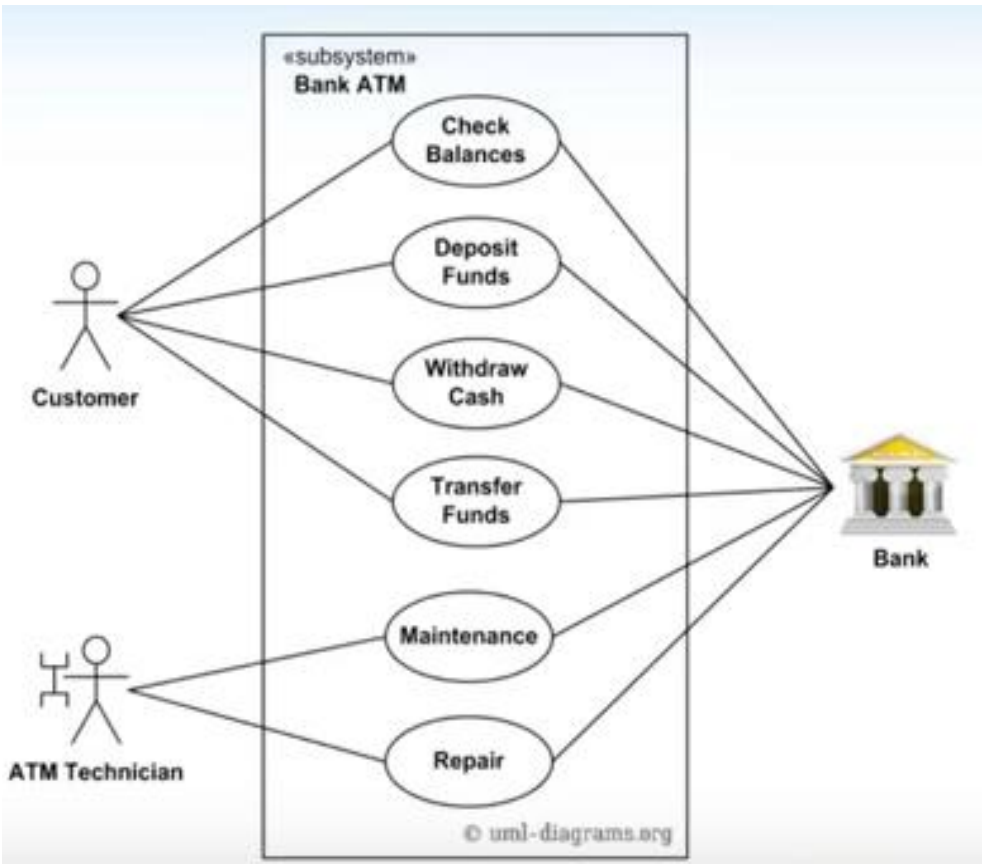


Use Case Diagram

د. فادي تركاوي

- **حالة الاستخدام (Use Case)**في هندسة البرمجيات وهندسة أنظمة هي وصف لخطوات أو اجراءات بين المستخدم ونظام البرمجيات الذي يقود المستخدم إلى شيء مفيد.

مثال

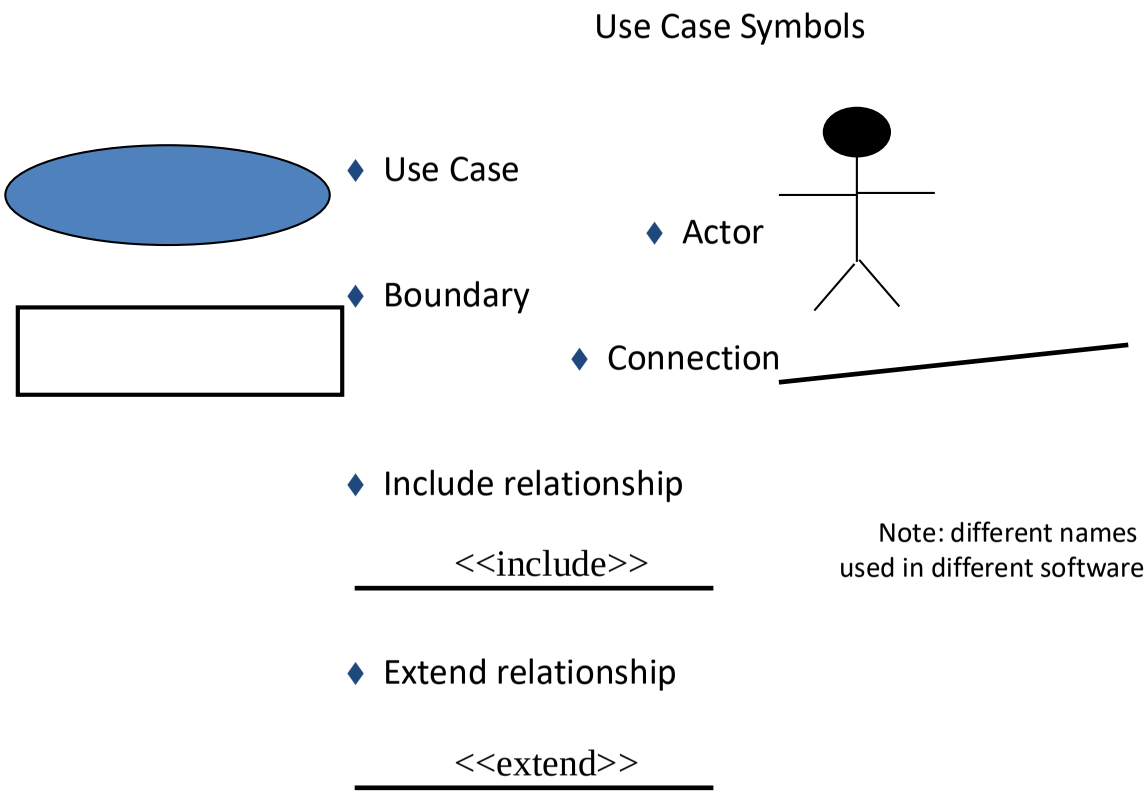


- Actors
- Use Cases
- Connectionis

ماذا تمثل use case ؟

- إذا Use cases تمثل وظائف النظام
- و هي تشير إلى:
  - سلوك النظام من خلال وظائفه
  - مجموعة من الفعاليات و التي تنتج مخارج outputs

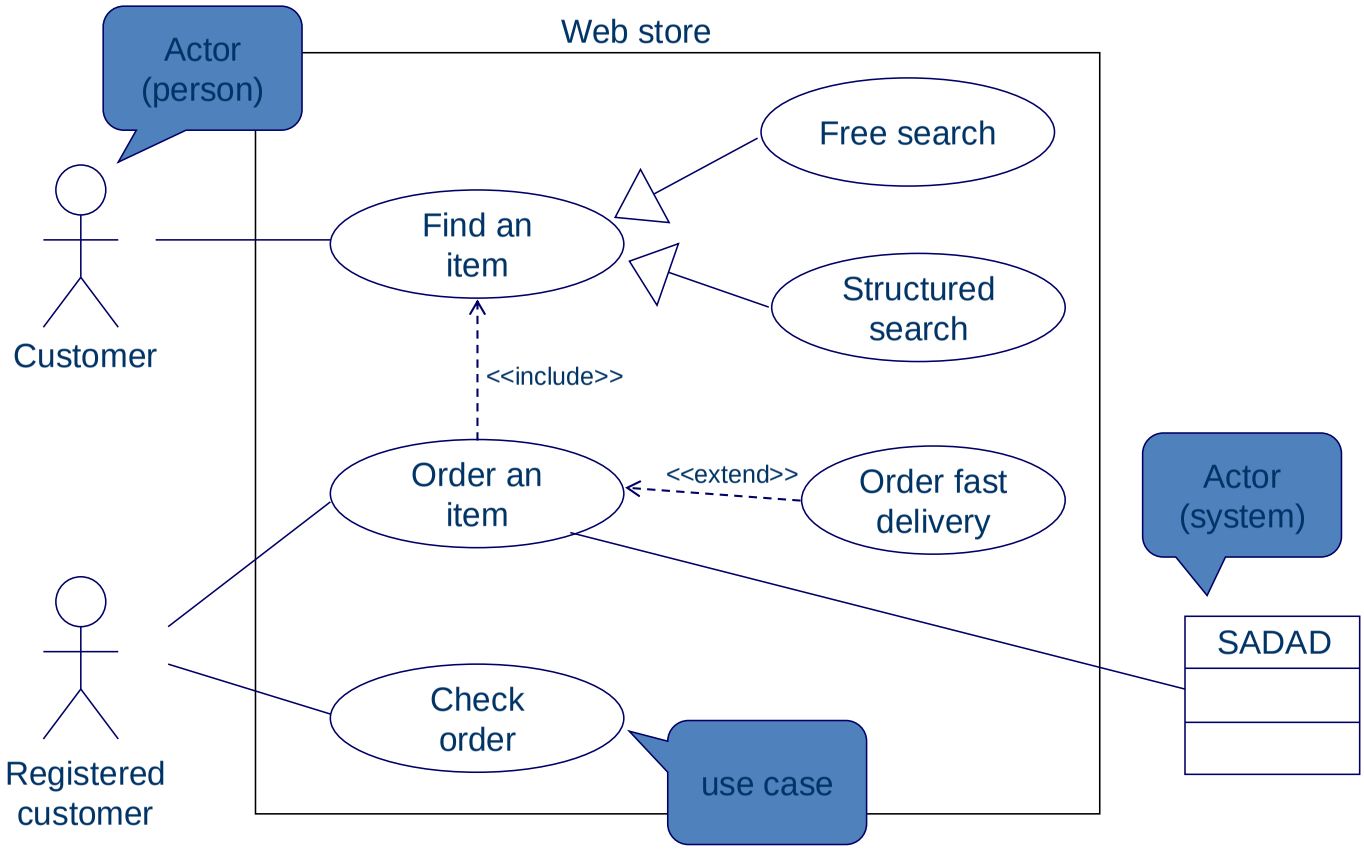
use case رموز



- **في المثال السابق يظهر الزبون كجهة فاعلة " "actorلأن الزبون هو الذي يستخدم نظام البنك . المخطط يأخذ الخطوات البسيطة المذكورة، و يظهرها على اساس إجراءات يمكن للعميل او الزبون ان ينفذها. ويمكن أيضا أن يقوم بسحب ، ايداع .... في مخطط حالة الاستخدام " Use Case Diagram" أيضا يتفاعل مع نظام.**

- **النظام بحاجة إلى أن يكون قادرا على تنفيذ الإجراءات لكافة حالات الاستخدام " "use casesالمذكورة. كلما تقدم المشروع قد تظهر حالات استخدام أخرى. و ربما يحتاج الزبون الى ان يضيف طلب اخر الى الطلبات التي وضعها سابقا. يمكن بسهولة توسيع هذا المخطط الى مدى وصف كامل لنظام**

مثال عن استخدام use case



Relationship

- Genralization في حال كان الجواب هل هذه الحالة هي حالة من النوع العام لنوع أخر
- Extend تمثل اعتماد خيار مخصص مثلا على حالة استخدام عامة.
- بينما include تمثل اعتماد حالة على حالة أخرى. و اتجاه السهم يمثل حالة ا لاعتماد.

Linking Use Cases

- *Association* relationships
- *Generalization* relationships
  - One element (child) "is based on" another element (parent)
- *Include* relationships
  - One use case (base) includes the functionality of another (inclusion case)
  - Supports re-use of functionality
- *Extend* relationships
  - One use case (extension) extends the behavior of another (base)

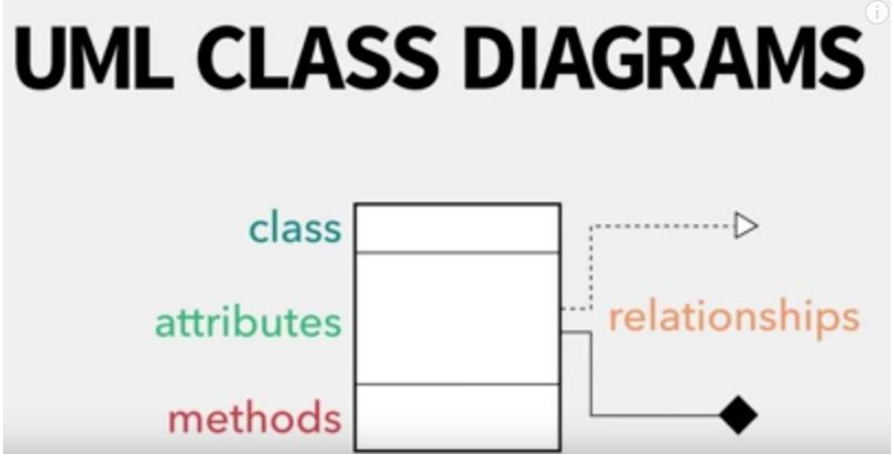
مخطط الفئة

Class Diagram

د. فادي تركاوي

تعريف Class Diagram

- مخطط الفئات (classes)هو جزءٌ مهمٌ جدًا من لغة النمذجة الموحدة UML، وهو مخطط هيكلي مهمته عرض الفئات بنظامٍ معيّن مع جميع العلاقات التي تربط بينها، وهو أشهر نوع من المخططات في هندسة البرمجيات.
- توفير وصف بشكل مستقل عن التطبيق للأنواع التي يتم استخدامها في نظام و يتم تمريرها بين مكوناته.



الفئات classes

- تتألف الفئات من ثلاثة أشياء : اسم " "nameوالصفات ""attributes"، والعمليات " ."operationsهذا مثال على الفئة " "classأدناه.

نوع البيانات

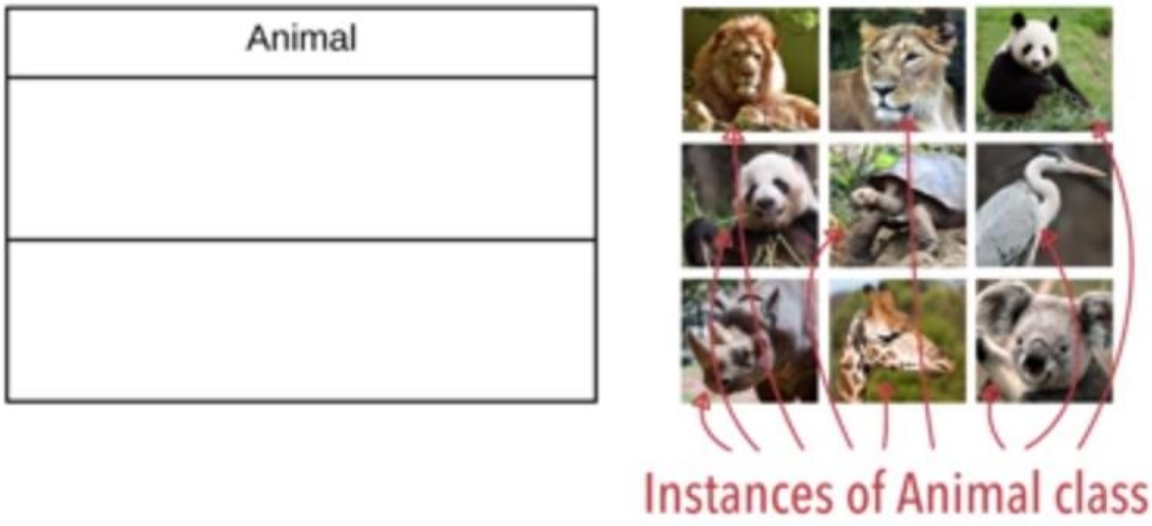
visibility attribute-name : Type

visibility operation-name (parameter1 : Type1, ...) : Type

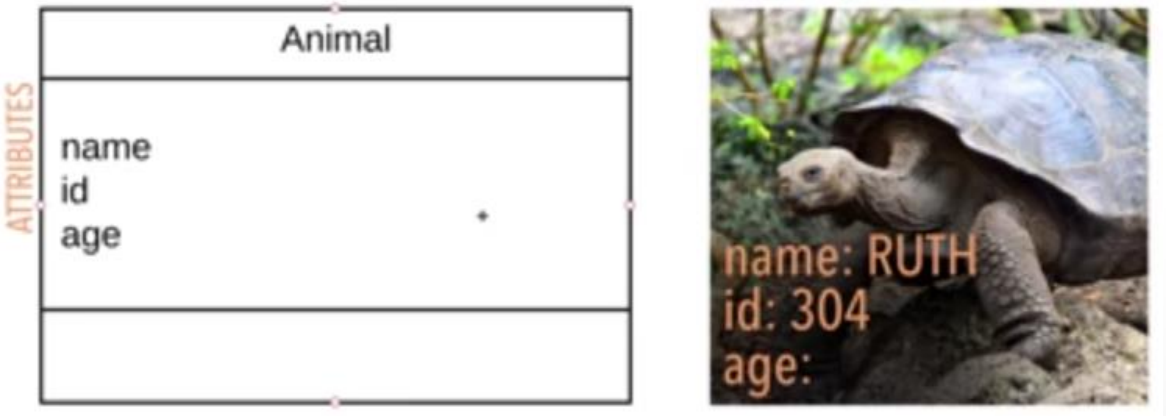
+AddItem (item : MenuItem, quantity : Integer) : Boolean

مثال

Zoo System

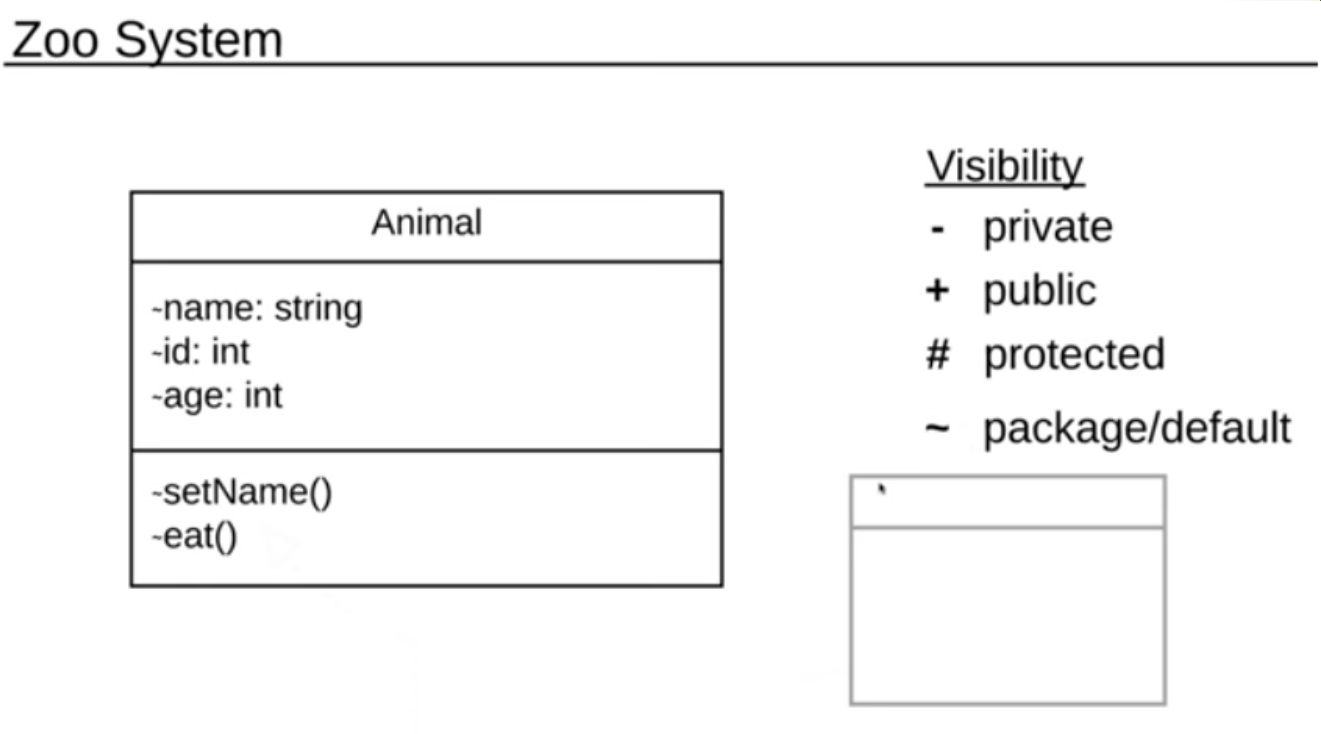


مثال تتمة

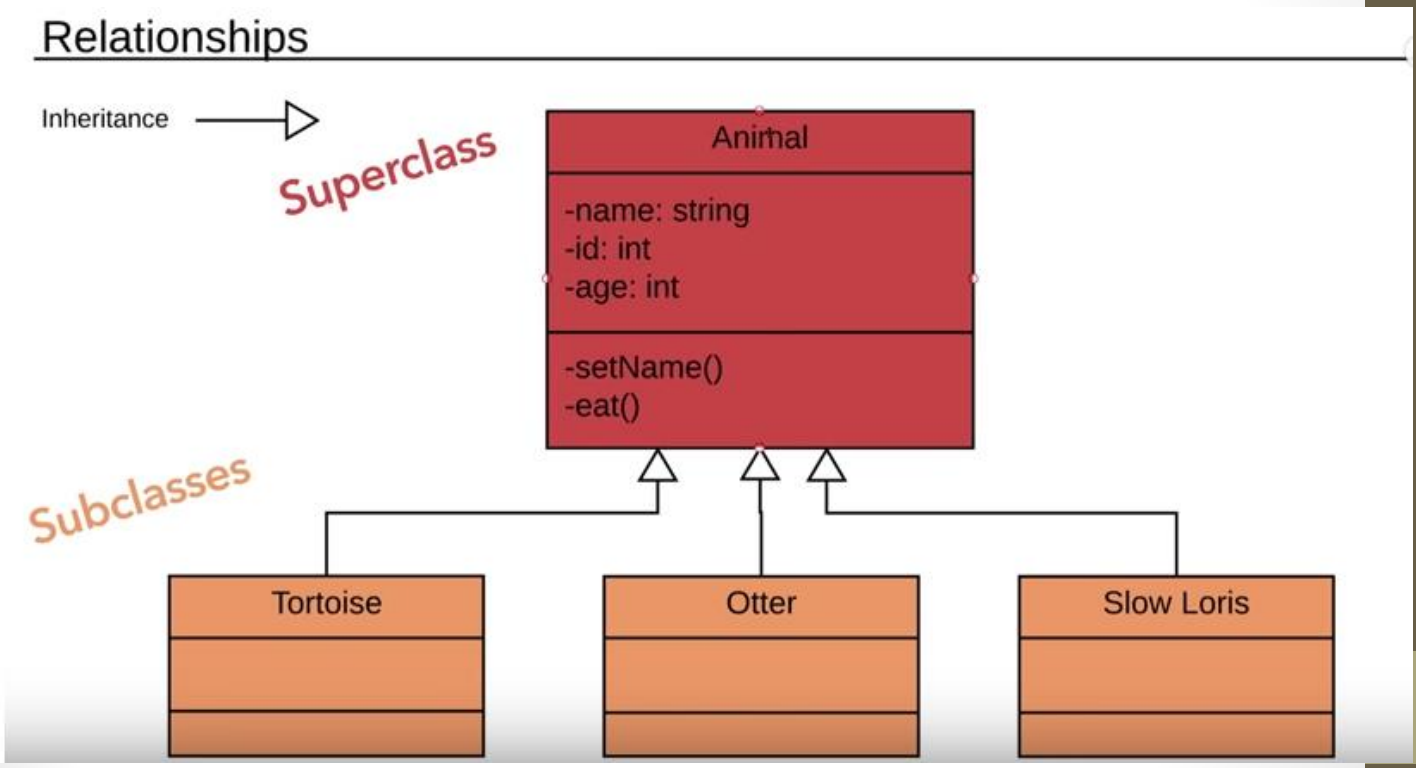


Visibility

المعنى	الشكل المختصر	الاسم
يمكن الوصول إليها من كافة الأنواع الأخرى.	+	Public
يمكن الوصول إليها فقط من التعريف الداخلي لهذا النوع.	-	Private
يمكن الوصول إليها فقط ضمن الحزمة التي تتضمن هذا النوع، وفي أي حزم تقوم باستيرادها بشكل صريح. راجع تعريف مساحات الأسماء و الحزم.	~	الحزمة
يمكن الوصول إليها فقط من هذا النوع و الأنواع التي ترث منه. راجع الوراثة .	#	محمي

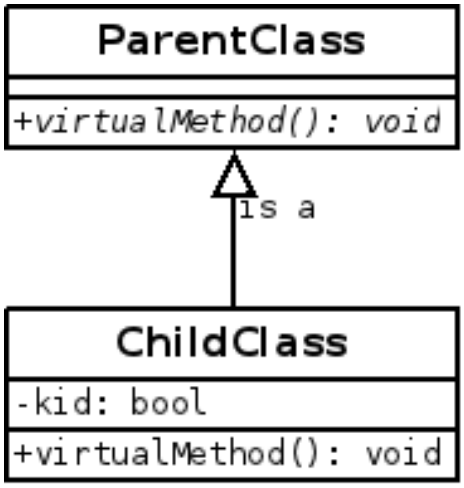


Relationship العلاقات



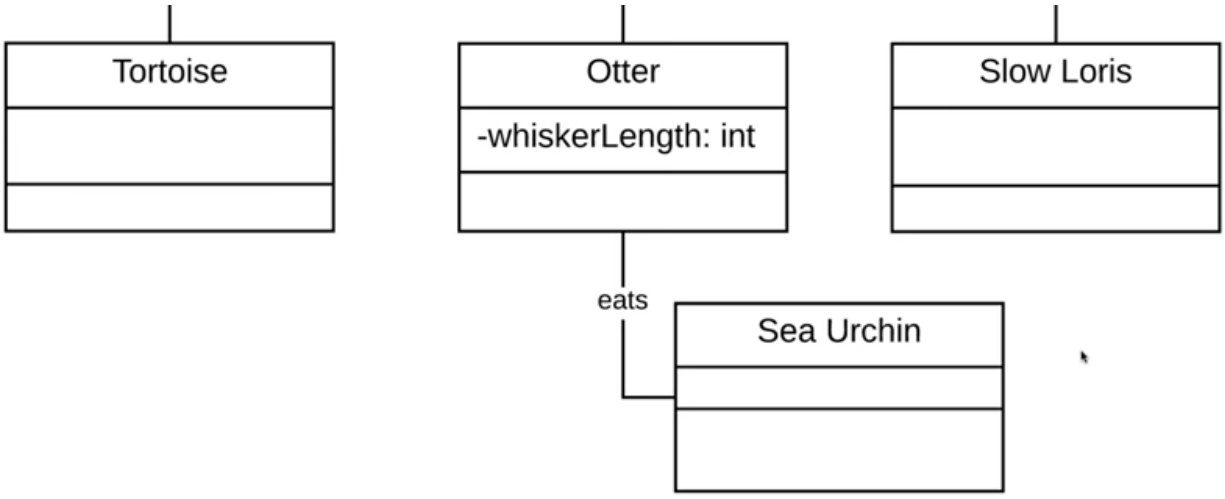
الوراثة

- وراثة الفئات -في ما يتعلق بمخططات UML- هي علاقة تعميم (generalization) التي تمثل علاقة "هو" (is a) على مستوى الفئة، المخطط الآتي يُظهِر كيفية رسم التعميم.



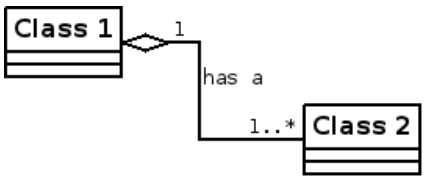
Association – الارتباط

- شكل آخر من أشكال العلاقات في مخططات الفئات هو الارتباط ((association، وهو علاقة على مستوى الكائنات ((object-levelأي أنه يحدث بين كائناتٍ لأصنافٍ مرتبطةٍ لا يوجد اعتماد مجرد ارتباط ؛ لذا تُمثَّل كل العلاقة كعائلة من الوصلات (.(linksهنالك عدة أنواع من الارتباط مُحدَّدة أكثر (التجميع aggregationو التألف .(composition



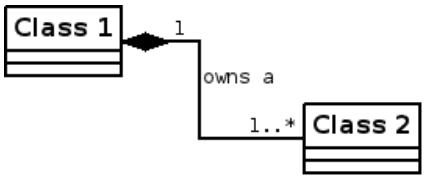
Aggregation – التجميع

- التجميع هو شكل أكثر تحديدًا وتخصيصًا من الارتباط. وهو علاقة "لديه" (has a)؛ التمثيل الرسومي لهذه العلاقة هو الآتي:



Composition – التألف

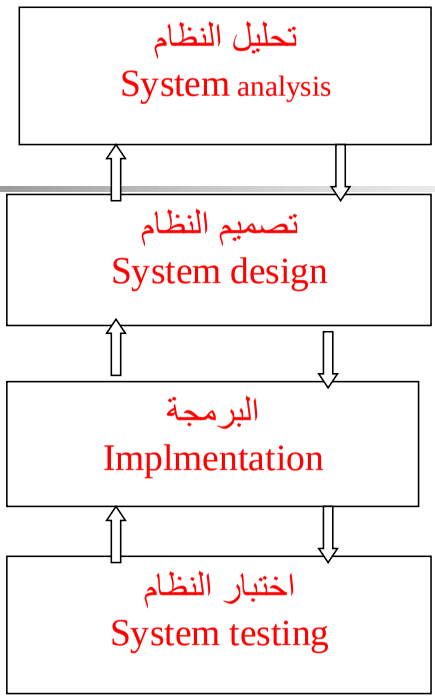
- شكل أكثر تخصيصًا من التجميع هو التألف ((compositionفبدلًا من علاقة "لديه" ((has aتكون العلاقة هي "يملك" (.(owns aوهذا ملائمٌ للعلاقات التي لا يمكن أن يتواجد فيها كائن إلا كجزءٍ من كائنٍ آخر. على سبيل المثال، إن كنت هنالك طائرة تملك جناحًا فهذا تألف، فماذا ستفعل بالجناح لوحده؟ لكن إن كانت هنالك بركة فيها بعض البط فهذا تجميع، لأنه يملك للبط أن يعيش دون بركة (وإن لم يكن سعيدًا بذلك)، والبركة ستبقى بركة حتى لو لم يكن فيها بط؛ التمثيل الرسومي لعملية التألف هو مثل التجميع، لكن المُعيَّن مملوء وليس مُفرَّغ.



دورة حياة نظم المعلومات

دورة حياة نظم المعلومات

- كثير ما يعاني نظم المعلومات من مشاكل مستمرة تتطلب إما تطوير النظام أو إحلاله بنظام جديد وهذا الإحلال أو التطوير يتم بعد إجراء حلقه متصلة من الدراسات تشمل عدة مراحل. كما في الشكل التالي: -



تحليل النظام system analysis

يتم تحليل النظام من خلال مرحلتين

- اولاً:- تعريف المشكلة **Problem Definition**

يجب على المحلل أن يفهم المشكلة فهما عميقا ويجب أن تكون واضحة أمامه تماما وان يصنف هذه المشاكل إلى أنواع مشاكل معروفه وهي كتالي:

- *مشكلة الثقة* **The Problem Of Reliability**
- من خصائص المشكلة والتي توجد في النظام الحالي المؤسس أن الإجراءات تعمل بعض الوقت ولا تستطيع أن تعمل طول الوقت وكذلك فان نفس الإجراء يعطي نتائج مختلفة من وقت لأخر و بالتالي يسبب عدم الثقة في إجراءات النظام الحالي .
- *مشكلة صحة النتائج* **The Problem Of Validity**
- هذه المشكلة تكمن في أن النظام يخرج أحيانا نتائج كثيرة غير واضحة وغير صحيحة وبالتالي تسبب اضطرابات للمستخدمين.

- **The Problem Of Accuracy** *مشكلة الدقة*

- هذه المشكلة تكون معظمها معتمده على صحة البيانات فان صحت البيانات صحت نتائج النظام وان خطأت أصبح النظام غير الدقيق.

- **The Problem Of Capacity** *مشكلة القدرة أو الاستيعاب*
  - قدرة التشغيل غير كافيه
  - قدرة التخزين غير كافيه
  - قدرة العاملين غير كافيه.

- **The Problem Of Throughput** *مشكلة الانجازأو الإنتاج*

- هذه المشكلة ترتبط بإنتاجيه وقدرة النظام فإذا كان النظام قدرته عاليه و الانتاجيه منخفضة فهذا معناه وجود

ثانياً: - تحليل متطلبات النظام System requirements analysis

- في هذه الخطوة يقوم محلل النظام system analystبالاجتماع مع مستخدم لنظام(user)للتعرف على مايحتاجه المستخدم من تعديل النظام القديم أو إحلاله بنظام أخر.

Developing Information systems
And
The system Developing live cycle

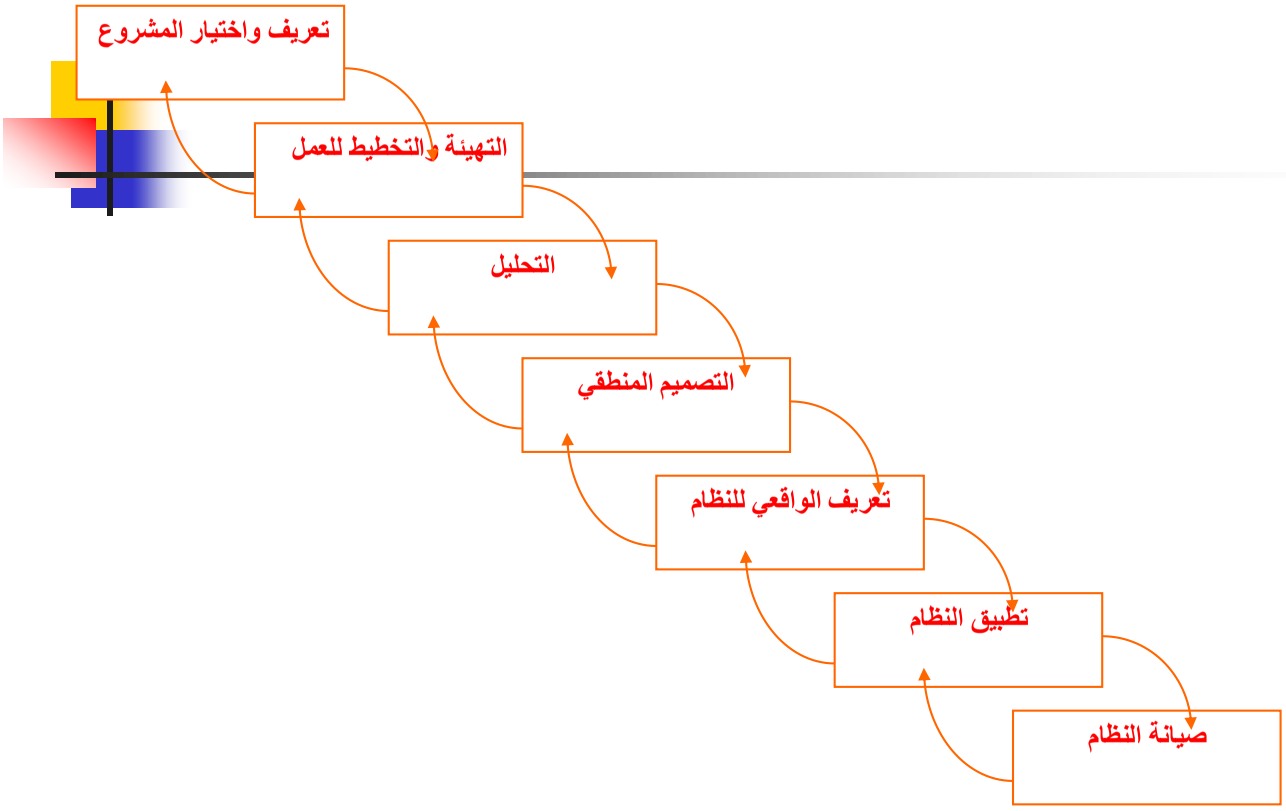
- **منهجية تطوير النظم:**

عملية رتيبة تتبع في المنظمة وذلك لتوصيل جميع الخطوات الضرورية على التحليل والتصميم ثم البرمجة وصيانة نظام المعلومات.

دورة حياة تطوير النظام SDLC

- وهي عبارة عن منهجية تقليدية تستخدم لتطوير وصيانة واستبدال أنظمة المعلومات.

وهذه الدورة مكونة من سبع مراحل يمر بها النظام من بداية في التفكير في إنشاء مشروع إلى مراحل صيانة المشروع كما سوف نشاهد في الشكل التالي وهذه المراحل متسلسلة كما:



دورة حياة النظام SDLC

١. تعريف واختيار المشروع

Project Identification And Selection :

هذه المرحلة تعتبر المرحلة الأولى من دورة حياة النظام
وخلال هذه المرحلة يتم التعرف على احتياجات نظام المنظمة وتحليلها و وضع أولويات لها وترتيبها.

الناتج من هذه المرحلة: system service request (SSR)

وهو عبارة عن ملخص المشكلة

دورة حياة النظام SDLC

٢. التهيئة والتخطيط للعمل
: Project Identification And planning

تعتبر المرحلة الثانية من دورة حياة النظام ومن خلالها يتم شرح نظام المعلومات المتوقع تنفيذه..

وهل من الأفضل الاستمرار في إكمال العمل في المشروع (النظام) أو إلغاءه.

تخطيط وتهيئة المشروع ومن ثم نقرر دراسة الجدوى من هذا المشروع واتخاذ قرار المتابعة

دورة حياة النظام SDLC

- أهم الخطوات لدراسة الجدوى للمشروع :

١- هل المشروع مربح اقتصادياً ٢- فنياً

٣- عملياً

٤- هل الوقت كافي لعمل المشروع ٥- قانونياً ٦- سياسياً

وكذلك يتم اعتماد الخطة لعمل الأجزاء المتبقية من **SDLC** في حال اتخاذ القرار بالاستمرار في المشروع وعدم إلغاءه.

الناتج من هذه المرحلة : Plan ,detailed problem definition ,cost/benefit analysis

دورة حياة النظام SDLC

٣. التحليل Analysis:

وهذه هي المرحلة الثالثة من **SDLC** ويتم خلالها دراسة النظام الحالي واقتراح النظم البديلة التي سوف تحل محله.

الناتج من هذه المرحلة : Requirement structure

دورة حياة النظام SDLC

٤. التصميم المنطقي Logical Design:

وهذه هي المرحلة الرابعة من **SDLC** ويتم خلالها وصف لكل وظائف النظام الذي وقع عليه الاختيار من بين الأنظمة البديلة وهذا الوصف لا يكون مرتبط بأي نوع من لغات البرمجة أو تطبيقات الحاسب الآلي.

الناتج من هذه المرحلة Logical Data model :

دورة حياة النظام SDLC

٥. التصميم الواقعي للنظام Physical Design:

- وهذه هي الخطوة الخامسة من **SDLC** وفيه يتم تحويل التصميم المنطقي إلى تفاصيل ترتبط بالتقنية التي سوف تتبع (لغات البرمجة وتطبيقات الحاسب) وخلال هذه المرحلة يمكن إنجاز كل مهام البرمجة وتطبيقات الحاسب.
- الناتج من هذه المرحلة :

Physical Data model

دورة حياة النظام SDLC

٦. تطبيق النظام implementation:

- وهذه هي الخطوة السادسة من **SDLC** وفيها نبدأ بكتابة الـ **code** للنظام بعد اختبار لغة البرمجة في التصميم الفعلي للنظام وبعد الانتهاء من كتابة الـ **code** نقوم بعملية اختبار النظم وذلك بإدخال بيانات حقيقية لمعرفة نقاط الضعف في النظام ومن ثم تعديلها.
- ثم نقوم بعملية تحميل **-installation-** للنظام على أجهزة المنظمة ، وبعد ذلك نقوم بعملية الدعم الفني للمستخدمين.
- الناتج من هذه المرحلة : Source Code

دورة حياة النظام SDLC

٧. صيانة النظام Maintenance:

وهذه هي الخطوة السابعة والأخيرة من **SDLC** وفيها يتم إجراء التعديلات على النظم بناءً على طلب المستخدمين.

أنواع دورة حياة تطوير النظم
Type of System Developing Live Cycle

١- دورة حياة تطوير النظام التقليدية Traditional SDLC:

عيوب الطريقة التقليدية:

- من الصعب العودة إلى مرحلة سابقة بمجرد الشروع في المرحلة التي تليها.
- يعتبر وقت التحليل والتصميم قليلاً مقارنة بالوقت المحدد للبرمجة.

أنواع دورة حياة تطوير النظم
Type of System Developing Live Cycle

١ـ التحليل المركب والتصميم البنائي
Structured Analysis And Structure Design

* يستخدم تقنيات **DFD** الهدف منها هو تقليل الوقت والجهد.
* من السهل العودة إلى المرحلة السابقة في حال تغير المتطلبات.
* يتم فيه تقسيم العملية (المشكلة) إلى وحدات اصغر يكون من السهل إدارتها والسيطرة عليها.

أنواع دورة حياة تطوير النظم
Type of System Developing Live Cycle

٢.التحليل والتصميم الشيئي Object Oriented Approach:

* مبني على أساس الأهداف العامة للمنطقة وليس على نوع البيانات أو طرف المعالجة.

<u>* يتميز بثلاث سمات هي:</u>

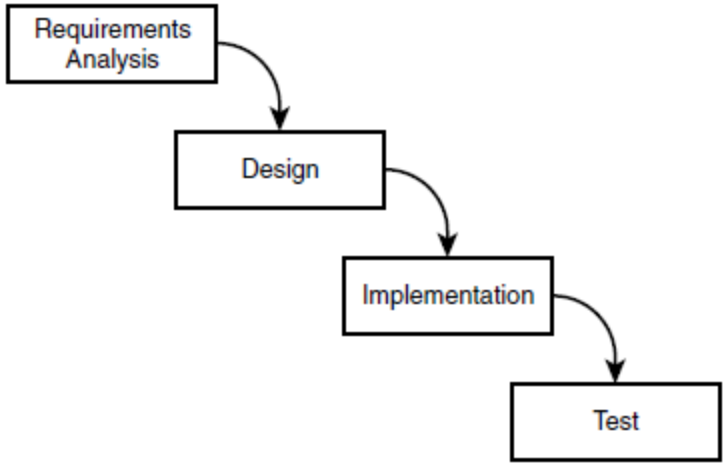
١- الوراثة **Inheritance**.

٢- التغليف أو الاحتواء **Encapsulate**.

٣- تعدد الأشكال **Polymorphism**

نموذج الشلال(**Waterfall Model**)

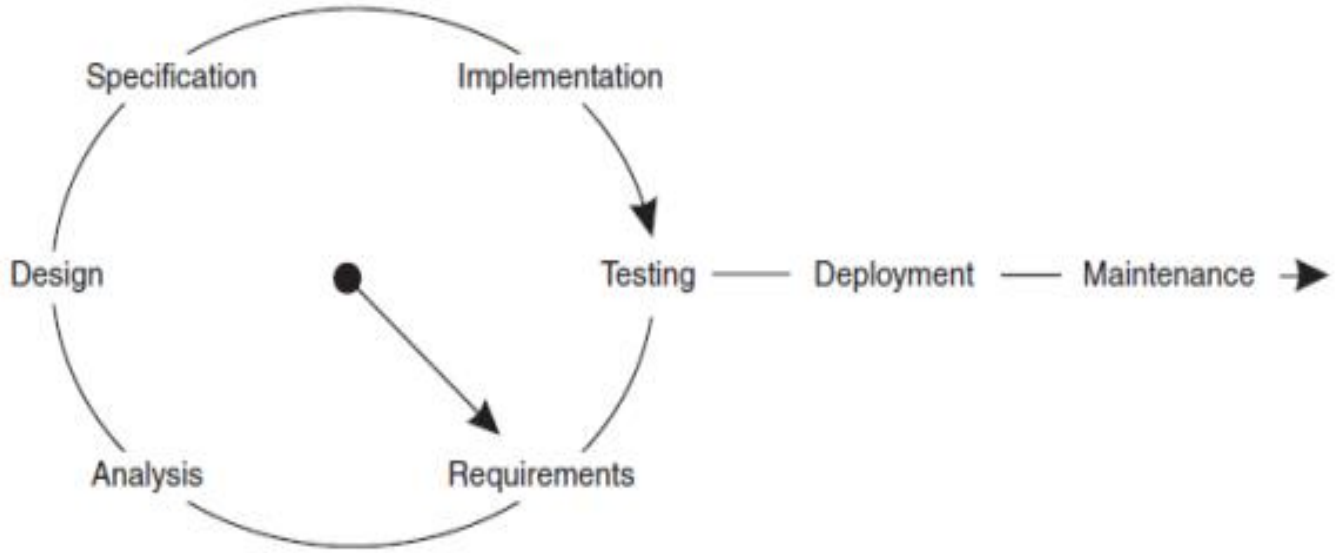
- مر معنا في المحاضرة الماضية دورة تطوير حياة النظام
- معرفة وتحليل المشكلة والمتطلبات وتسمى.(**Analysis**)
- طرح حلول المشكلة وتسمى.(**Design**)
- تنفيذ التصميم ويسمى.(**Implementation**)
- إجراء إختبار ويسمى testing

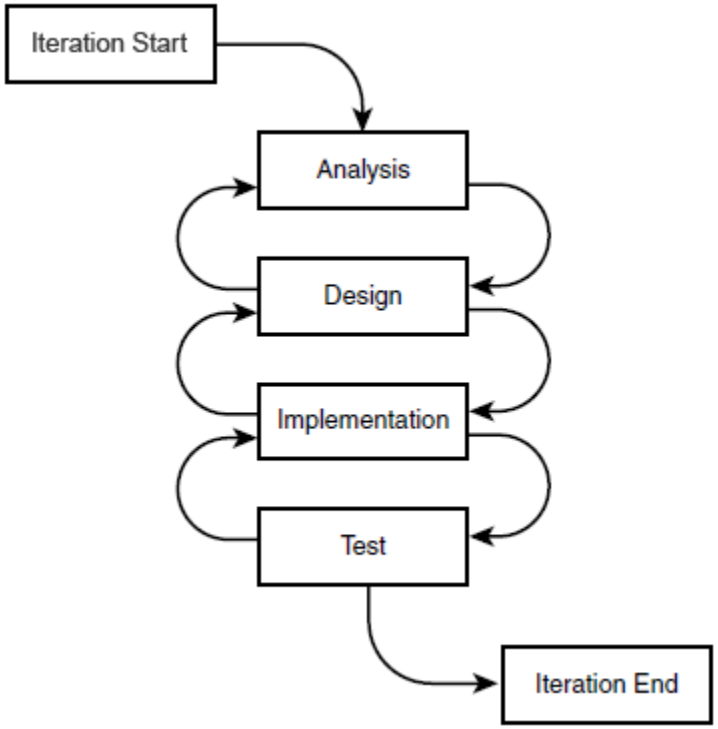


- **العيوب:**
لأن هذة العملية تكون بالترتيب بشكل متنازل فإنها لاتصلح إلا في المشاريع الصغيرة والتي لا يوجد فيها خطورة والسبب أننا لا نستطيع العوده للخلف بتاتاً
فمثلاً:
عندما يبدأ فريق العمل في مشروع معين يقوم المحلل أولاً بتحليل البرنامج بالكامل وكتابة جميع الوثائق المطلوبة ومن ثم يمرر هذة الوثائق للمصمم الذي يقوم بتصميم النظام ووضع حلوله أيضاً ، ومن ثم تمر بالمبرمج الذي يقوم بكتابة الكود وعندما ينتهي تجرى الأختبارات على هذا البرنامج للتأكد من عدم وجود أخطاء وأخيراً يسلم الى الزبون
السؤال هنا:
ماذا لو تغيرت متطلبات هذا الزبون بعد مرور ٣ أشهر من التحليل والتصميم وبالبدء بالكود ، كارثه!!
وحتى لو أن المتطلبات لم تتغير فإن وجود خطأ في مرحلة من المراحل يؤدي إلى دمار كل المشروع!!
- **الحل:**
الحل أنهم قاموا بتطوير نموذج الشلال (**Waterfall Model**) إلى نموذج جديد يسمى النموذج الحلزوني(**Spiral Model**)

النموذج الحلزوني(Spiral Model)

- أخذ المتطلبات من الزبون(**Requirements**)
- تحليل جزء من المشكلة وتسمى.(**Analysis**)
- طرح حلول المشكلة وتسمى.(**Design**)
- التأكد من المواصفات(**Specification**)
- تنفيذ التصميم ويسمى.(**Implementation**)
- إجراء إختبار ويسمى.(**Testing**)
- نشر هذا الجزء من البرنامج ويسمى.(**Deployment**)
- صيانة الأخطاء التي تظهر فيه وتسمى.(**Maintenance**)


Specification
Implementation
Design
Testing
Deployment
Maintenance
Analysis
Requirements


Iteration Start
Analysis
Design
Implementation
Test
Iteration End

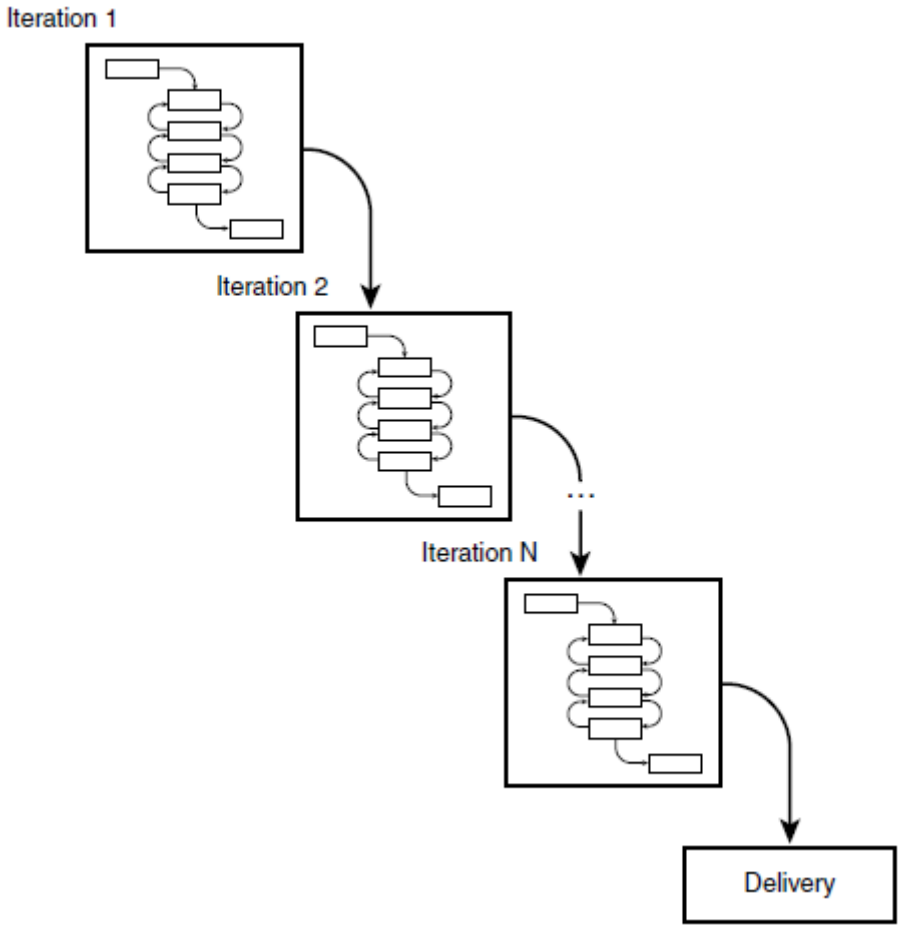
- **العيوب:**

نعم لقد أصبحت عملية التطوير هي عباره عن عدة دورات كل منها تستطيع التحرك للأمام أو للخلف حسب ما تريده ولكن ما زالت المشاكل بالظهور حيث الدورات السابقه كانت لإصلاح خلل وغالباً ما تكون ٤ دورات فقط!!!

- **الحل:**

نريد تقسيم المشروع لأكثر من دوره وفي كل مره نقوم بتطوير جزئيه معينه ولهذا جمعوا بين جميع هذه الطرق (الشلال ، الحلزوني ، المتكرر) كل منها توجد فيه خاصيه مفيده في عمليه التطوير ولذا يمكن أن نجمع كل هذه الطرق مع بعضها البعض وبالتالي نستفيد من جميع المزايا في كل واحدة من الطرق

- **الطريقة ٤ :**الجمع بين النماذج الشلال والحلزوني والمتكرر **(Waterfall, Spiral, Iterative Models)**:


Iteration 1
Iteration 2
...
Iteration N
Delivery

- تعدّ البرمجيّات العنصر الرئيسي في العديد من الأنظمة والأجهزة المنتشرة في حياتنا، وعلى الرغم من تأثير العديد من العوامل على وثوقيّة هذه البرمجيّات ونجاحها كالتصميم والتنفيذ الدقيقين، إلّا أنّ الاختبار هو الوسيلة الرئيسيّة التي تعتمدها الصناعة لتقييم البرمجيات التي لا تزال قيد التطوير

- بدايةً، تمكّنت الشركات البرمجية منذ فترةٍ ليست بطويلةٍ من تحمّل تكاليف توظيف مبرمجين لا يجيدون إجراء أيّ اختبارٍ لمشاريعهم، بالإضافة إلى مختبرين لا يعرفون أيّ شيءٍ عن البرمجة. ولم يكن من الضروري لصنّاع البرمجيات أن يعرف موظّفوهم المبادئ التقنيّة للاختبار أو حتّى تطوير البرمجيات. وبذلك كان اختبار البرمجيات في بداياته نشاطاً غيرَ تقنيّ ويتمّ النظر إليه على أنّه عملٌ إداري.

- *ومع تطوّر هندسة البرمجيّات وانتشار برمجيّاتها في حياتنا اليوميّة، ظهرت متطلّباتٌ صارمةٌ بخصوص وثوقيّة*

ما هي اختبارات جودة البرمجيّات؟

هذه البرمجيّات وصيانتها وأمنها. وكان لا بدّ للصناعة أن تتجاوب مع هذه المتطلّبات الجديدة بتطوير طرق اختبارها لهذه البرمجيّات، ممّا يعني زيادة الخبرة التقنيّة لمهندسي الاختبار بالإضافة إلى تعزيز أهميّة الاختبار وضرورته عند المبرمجين.

- *تدعى عمليّة اختبار البرمجيات بـ:*

. control quality softwareمراقبة جودة البرمجيات

. assurance quality softwareضمان جودة البرمجيات

- الخطأ :Errorوهو أي خطأ قام به المهندس كنتيجةٍ عن سوء الفهم للمتطلّبات أو التصميم.
العيب :Faultوهو ظهور هذا الخطأ ضمن الكود، وندعوه غالباً) Bugبمعنى إصابة.(
الفشل :Failureوهو السلوك أو النتيجة غير الصحيحة والناتجة عن تنفيذ كود يحوي عيباً ما،
قد يكون هذا الفشل مباشراً)كأن ينهار النظام(أو قد يظهر لاحقاً خلال التنفيذ

- هل يكون النظام خالياً من العيوب إن تجاوز كلّ اختباراته بنجاح؟
للأسف لا، فقد تكون العيوب مختبئة ضمن أجزاءٍ نادرة التنفيذ من الكود، وبذلك تكون وظيفة الاختبار إثبات وجود العيوب وليس عدم وجودها وتأثيرها على النظام.
ولكن في حال قمنا بتشكيل مجموعة اختباراتٍ واسعةٍ تشمل كافة وظائف النظام، فاجتياز هذه المجموعة يعني أنّ نظامنا ذو جودةٍ عاليةٍ وقابلٌ للنشر.

١ .تجميع كافّة وظائف وسلوكيّات النظام على اعتبار أنّها فضاء واسع.
٢ .تقسيم هذا الفضاء إلى أجزاء مختلفة الوظيفة)أو التنفيذ.(
٣تكوين اختباراتٍ تماثل نتيجة كلّ جزءٍ من هذا الفضاء وتنفيذ هذه الاختبارات

مثال

- لنفرض أنّه لدينا برنامجٌ يقوم بإيجاد القاسم المشترك الأكبر لعددين xو y، فمن الممكن أن تكون أجزاء
الفضاء في هذه الحالة:
اختبار حالةٍ شائعة مثل ٦ =xو) ٩ =yالقاسم المشترك الأكبر هنا= .(٣
اختبار كَون أحد العددين هو القاسم المشترك الأكبر مثل ٢ =xو)
٤ =yالقاسم المشترك الأكبر= .(٢
اختبار عددين أوليّين مثل ٣ =xو) ٥ =yالقاسم المشترك الأكبر= .(١
اختبار كون أحد العددين مساوياً للصفر مثل ٩ =xو) ٠ =yلا يوجد قاسم مشترك أكبر.(
اختبار كون أحد العددين سالباً مثل ٣ -=xو) ٩ =yالقاسم المشترك الأكبر هنا هو العدد السالب.(
ويتضّح لنا من المثال السابق أنّ اختبارَ نظامٍ ما بالكامل هو أمرٌ مستحيل

أنواع الاختبارات

١. اختبار الوحدة :Testing Unitويهتمّ بالمفاهيم منخفضة المستوى من النظام، إذ يَجري الاختبار هنا ضمن الكود البرمجي ومن أجل كلّ نموذج وكلّ عمليّة.
مثال: من أجل كلّ تابعٍ أو إجرائيّةٍ ()foo سنَجد تابعاً لاختباره يدعى ()testFoo
٢. اختبار التكامل :Testing Integrationويهتمّ بضمان عمل جميع الأجزاء بشكلٍ صحيحٍ بعد دمجها معاً وخاصّةً عندما يعمل أكثرُ من مطوّرٍ واحدٍ على كتابة الكود.
٣. اختبار النظام :Testing Systemللتأكّد من عمل جميع الوظائف الكبيرة ضمن النظام وتطبيق جميع متطلّبات الزبون.
٤. اختبار القبول :Testing Acceptanceويجري عادةً من قبل المستخدمين للتأكّد من تلبية النظام لاحتياجاتهم.
٥. - اختبارات الكود:
-١اختبار الصندوق الأسود :Testing Box Blackهل سلوك النظام مطابقٌ لما جاء في المتطلّبات؟
-٢اختبار الصندوق الرمادي :Testing Box Greyهل سلوك النظام مطابقٌ لما جاء في المتطلبات مع وجود معرفة ضئيلة ببنية هذا النظام.
-٣اختبار الصندوق الأبيض :Testing Box Whiteاختبار جميع أجزاء الكود من عباراتٍ وتفرّعاتٍ وغيرها.

٦. اختبار التراجع :Testing Regressionويقصد به إعادة اختبار البرمجيّة بعد أن تمّ تعديلها أو إصلاح أخطاء سبق وتمّ العثور عليها. ويتطلّب هذا النوع أكبر قدرٍ من الجهد وغالباً ما يتمّ إجراء العديد من جولات الاختبار التراجعيّة في الأنظمة الكبيرة، لأنّ أيَّ تعديلٍ بسيطٍ ضمن أحد أجزاء البرمجيّة قد يسبّب غالباً مشاكل في أماكن بعيدةٍ حتّى، وتكون وظيفة الاختبار التراجعيّ اكتشاف هذا النوع من الأخطاء.

٧. الاختبارات المؤتمتة :Testing Automated إن تطبيق جميع الاختبارات السابقة يدوياً)وخصوصاً اختبارات الكود(أمرٌ قد يكون في غاية الصعوبة، ولكن من حسن الحظ، هناك العديد من الأدوات التي تقوم بتغطية وتعقّب الكود بأكمله. وتقوم هذه الأنظمة المؤتمتة بتوليد تقارير لإظهار النسبة المئوية المُنجَزة من اختبار الكود، كما أنّ بعضها يظهر لنا العبارة البرمجية التي تخضع حاليّاً للاختبار

مداخل التحليل و التصميم:

١. التحليل والتصميم من أعلى الى اسفل

يهتم مصمم النظام أولا علي استيفاء احتياجات الإدارة العليا وبعد ذلك احتياجات المستويات الأخرى. هذا المدخل في التصميم يعطي الأولوية للإدارة العليا في تحديد مخرجات النظام ومنها يتم تحديد وظائف النظام والبرامج التطبيقية التي تعني بهذه الأهداف وبعد ذلك يتم استيفاء احتياجات المستويات الإدارية الأخرى من المعلومات مما لا يتعارض مع الأهداف العامة للإدارة العليا. يمتاز هذا المدخل بضمان مساندة وتأييد الإدارة العليا للنظام مع ضمان تحقيق الأهداف العامة .

٢. التحليل والتصميم من اسفل الى اعلى

يركز علي الوفاء بالاحتياجات الأساسية من المعلومات عند المستويات التشغيلية أولا متجها من أسفل إلي اعلي حتى الوفاء باحتياجات الإدارة العليا .

الأولوية تكون للمدراء الذين يتخذون القرارات اليوميه وهم المسئولين عن الأنشطة التطبيقية مثل المخزون والأجور والبيع والشراء ....الخ .

هذا المدخل من أكثر المداخل استخداما في الحياة العملية نظرا لسهولة تحديد تنفيذ الاحتياجات الأساسية للمستويات التشغيلية ويتجه التحليل من أسفل إلي أعلي حتى ينتهي بنظام متكامل للمعلومات .

٣. التحليل والتصميم بالتجزئة والتجميع

يتم تجزئة النظام ككل إلي مجموعة وظائف ، وتبدأ عملية التصميم بالتركيز أولاً علي الوظائف الحساسة أي التي بها أعمال كتابية كثيرة وتتنوع فيها عمليات التشغيل – هذا المدخل مناسب للنظم الكبيرة والتي لا يمكن تصميم كل الوظائف والأنشطة في وقت واحد . يتم تصميم كل وظيفة بصفة مستقلة عن الوظائف الأخرى ثم محاولة الدمج بين هذه الوظائف وإيجاد التكامل بينها رغم أن عملية التكامل والتنسيق بين احتياجات الوظائف المختلفة في هذه الحالة والتي ستكون شاملة وتحتاج إلي مجهود كبير ويمكن التغلب علي هذه المشكلة بالتحديد الواضح للأهداف العامة للنظام ككل ثم الالتزام بهذه الأهداف عند تصميم الأنشطة الخاصة بالوظائف المختلفة في النظام.

محلل النظم:

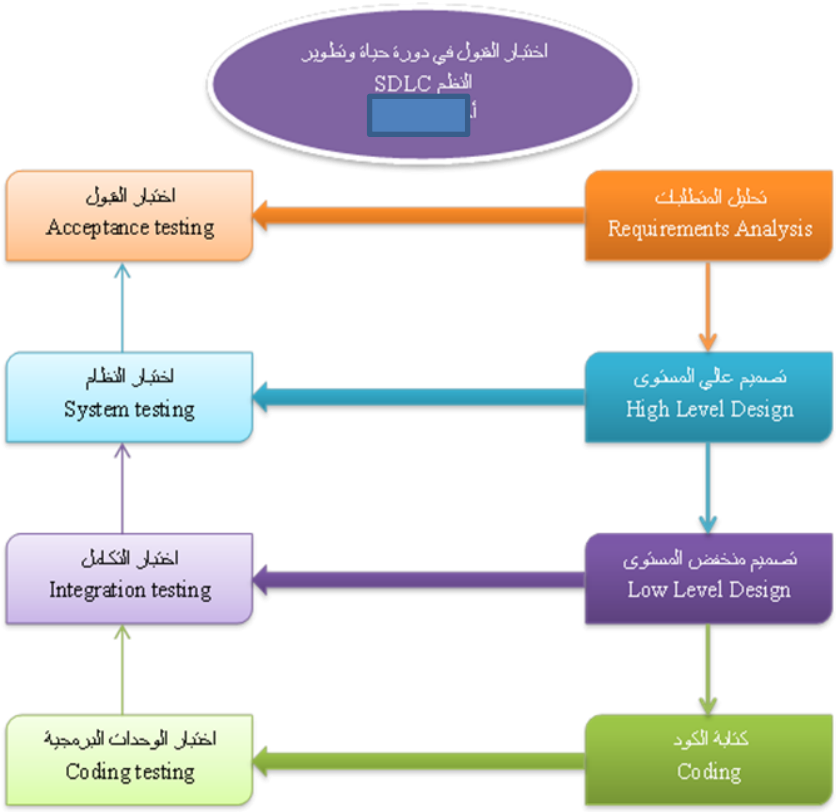
هو الشخص المسؤول عن دراسة النظام القائم بغرض تشخيص نقاط ضعفه ومشاكله ويقدم بعد ذلك تقريراً يتضمن مقترحات وحلولا مناسبة.

سمات المحلل الجيد:

- ✓ يكون مسئول عن تحليل النظام الجديد، وتصميمه و الإشراف على تشييده و التحويل و صيانته.
- ✓ يكون ماهرا في الاتصالات، و التحفيز و الإدارة.
- ✓ يكون قادرا على العمل كعضو أو كقائد الفريق .
- ✓ لدية مهارات حل مشاكل مرتفعة المستوى.
- ✓ القدرة على عرض الافكار الجديدة وإقناع الآخرين.
- ✓ القدرة على إتخاذ القرار في الوقت المناسب
- ✓ القدرة على النظر الى الموضوع من وجهات نظر مختلفة
- ✓ القدرة على الابداع
- ✓ القدرة على الصبر والتحمل
- ✓ الاطلاع الواسع والمستمر على اخر المستجدات في مجال المعلومات والتقنيات.

مهام محلل النظم :

- ✓ جمع المعلومات الكافية لدراسة النظام القائم وتحديد متطلبات النظام الجديد
- ✓ تحديد المشاكل ونقاط الضعف التي يعاني منها النظام القائم
- ✓ تطوير حلول للمشاكل الموجودة
- ✓ تحديد أهداف النظام الجديد
- ✓ تحديد الجدوى الاقتصادية والفنية للنظام الجديد
- ✓ تصميم النظام



اختبار القبول في دورة حياة تطوير النظم

الصورة التالية توضح خطوات اختبار القبول خلال دورة حياة تطوير النظم.
حالات اختبار القبول تنفذ باستخدام بيانات تجريبية(بيانات اختبار) اوباستخدام سكريبتات اختبارالقبول ويتم مقارنة النتائج مع تلك المتوقعة.

معايير القبول:-

- اكتمال الوظائف وصحتها Functional Correctness and Completeness
- تكامل البيانات "سلامة البيانات" Data Integrity.
- تحويل البيانات Data Conversion.
- سهولة الاستخدام Usability.
- الاداء Performance.
- التوقيت المناسب "سرعة انجاز المهمات" Timeliness.
- السرية والاتاحة او التوفر Confidentiality and Availability.
- قابلية التحميل والتحديث Installability and Upgradability.
- قابلية التطوير Scalability.
- التوثيق Documents.

وصف خطة اختبار القبول Acceptance Testing Plan

خطة اختبار القبول تحتوي على التالي.

١. المقدمة Introduction.
٢. تصنيف اختبار القبول Acceptance Testing Categories .
٣. بيئة التشغيل Operational Environment.
٤. رقم تعريف حالة الفحص Test Case ID.
٥. عنوان الاختبار او الفحص Test Title.
٦. هدف الاختبار Test Objective.
٧. اجراء الاختبار Test Procedure.
٨. جدولة الاختبار Test Schedule.
٩. الموارد Resources.

استنتاج فحص القبول تكون واحدة من الثلاثة التالية:-

١. قبول النظام كما سلم " بمعنى ان فحص المتطلبات كان مطابق مائة بالمائة لما اتفق عليه"
٢. قبول النظام بعد التعديلات المطلوبة.
٣. رفض القبول.

نتائج فحص القبول تكون في صورة تقرير اختبار القبول.

تقرير اختبار القبول Acceptance Testing Report

تقرير اختبار القبول يحتوي على التالي.

١. معد التقرير Report Identifier
٢. تلخيص النتائج Summary the result
٣. الاختلافات Variations
٤. التوصيات Recommendation
٥. قائمة بالاشياء التي يجب ان تنفذ "التعديلات المطلوبة"Summary To-Do List
٦. قرار الموافقة Approval decision