الكتاب الجامعي لمقرر- هندسة البرمجيات

إعداد أستاذ الذكاء الاصطناعي والأنظمة الذكية المشارك

الدكتور/موفق البراق

الفصل الاول

المقدمة

1-1-المقدمة:

ان الحضارة البشرية الحالية تقوم على التطور السريع في مجال علوم الحاسوب ، وتقنيات المعلومات والمعتمدان بشكل اساسي على البرمجيات، فالبرمجيات متواجدة في كل شيء وليس فقط على الحاسبات، فهي متواجدة بالمكاتب والمعامل الصغيرة والكبيرة ، وبالأجهزة المنزلية والسيارات والطائرات والمصاعد والتليفونات ولعب الأطفال، وفي مالا يحصى من أجزاء الماكينات والمعدات الصناعية والخدمية بأنواعها. في مجتمع كهذا يعتمد على البرمجيات إلى هذا القدر، فإن البرمجيات تتزايد بالاستخدام المجتمعي يوميا وبشكل كبير جدا .. وعواقب الأخطاء في البرمجيات التي بدأت تكثر في شكل فيروسات وجراثيم برمجية في الآونة الأخيرة قد تؤدي إلى كوارث كبيرة، مثل القرصنة على انظمة البنوك والشركات المؤتمتة، تحطم واخطاء الصواريخ والطائرات الذكية، انهيار شبكة التليفون، توقف نظام المراقبة الجوية عن العمل، وغيرها. وقد اثبتت الدراسات السابقة العديد من الانتكاسات البرمجية التي سجلت وعجلت كثيرا الى الاهتمام واهمية علم هندسة البرمجيات وعقدت عليه كل الأمال للتغلب على كل الانتكاسات والمشاكل البرمجية فوجدوا انه تكمن الاهمية لعلم هندسة البرمجيات بالمفهوم الهندسي الذي يساهم في تحقيق المعايير الهندسية.

- **الاهمية بالمفهوم الهندسي الذي يساهم في تحقيق المعايير الهندسية التالية :**

1. زيادة ضبط الجودة وضمانها في المنتج
2. تساهم في تحسين ضوابط وأسس علمية لتقييس المنتج وتسعيره
3. البحث عن إمكانيات التطوير والتحسين على المنتج البرمجي
4. المساهمة في المفاهيم الادارية في إدارة الإنتاج البرمجي
5. مطابقة المتطلبات المستخدمة في النظام مع مع مخرجاته
6. تسهيل عملية الصيانة والدعم الفني بعد تسليم النظام وتشغيله
7. زيادة الاعتمادية على المنتجات البرمجية.
8. التقليل من زمن التأخر في تسليم المنتج البرمجي
9. تقليل المخاطرة في صناعة البرمجيات الضخمة و بالتالي عدم التعرض لانتكاسات مالية كبيرة.

في بداية صناعة البرمجيات كان يستخدم مبدأ الـ code-test والذي كان عالما من الفوضى ونتائجه كانت تشكل عقبات عند تسليم المنتج – تطويره – تحديد سعره – كشف أخطاءه – تحديد وضبط وتيرة العمل في إنتاجه وكان المبرمجون يقومون بكتابة الكود البرمجي للنظام –وهم من يقومون بفحصه، وكانت تحدث العديد من المشاكل والعيوب من غير معالجات.

هذا ما ساهم في تطوير اساليب هندسية من أجل العمل بشكل منظم و وفق قواعد لحل هذه المشاكل والصعوبات وظهور الحاجة إلى ما يسمى تحليل وتصميم النظم للحد من المشاكل السابقة والتي كان اهمها:

1. التغيير في المتطلبات اثناء تطوير البرمجيات.
2. التغيير الكبير والسريع في المكونات المادية Hardware مما يتطلب تغيير دائم في المكونات البرمجية (البرمجيات)
3. قصر في عمر المنتج البرمجي للاستخدام وعدم امكانية المحافظة عليه أطول والذي ينعكس على قيمته.
4. التنافس الكبير في مجال تطوير البرمجيات.

5. طبيعة المنتجات البرمجية الغير مرئية.
6. العيوب والاخطاء المرافقة لتطوير البرمجيات وعدم ضمان خلوها بسهولة.
7. استمرارية مهاجمتها بكثرة من الهاكرز (القراصنة) والفيروسات.
8. المخاطر والانتكاسات الضخمة المرافقة لتطوير البرمجيات.

1-2- ما هي البرمجيات : What is A Software?

هي عبارة عن انظمة برمجية تحتوي على مجموعة من المكونات Components المترابطة في ما بينها
ويقصد هنا بمصطلح Software هو نظام متكامل مكون من مجموعة برامج
وقد يتضمن النظام البرمجي – Software أجزاء ميكانيكية – إلكترونية – كهربائية والتي يمكن تسميتها بـ Hardware
وعادة ما تكون المكونات Components للنظام معتمدة على مكونات اخرى ،وبالتالي يكون سلوك النظام ككل معتمدا على مكونات ولتي تكون مترابطة داخليا فيما بينها بشكل يعكس عمل كل النظام ، وكثيرا ما يقيم نظام ما من خلال مكوناته.

- ❖ البرمجيات بشكل عام يتم تطويرها او هندستها ولا يمكن تصنيعها.
- ❖ البرمجيات لا تبلى ولا تتأكل مع الزمن بل انها تحتاج للتحديث والتغير بشكل مستمر.

تعاريف هندسة البرمجيات؟ Software Engineering Definitions وضع لها الكثير من المختصين في تطير البرمجيات ونظم المعلومات تعاريف عدة نذكر منها التعريفين التاليين:

1- هندسة البرمجيات هي وضع واستخدام مبادئ هندسية صحيحة للحصول على برمجيات موثوقة وتعمل بكفاءة على حواسيب حقيقية.

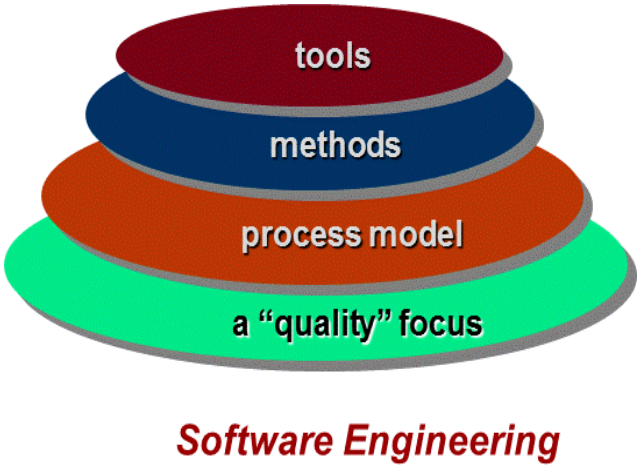
لقد عرفت المنظمة الدولية IEEE تعريفاً أشمل هو:

2- هندسة البرمجيات هي تطبيق ممنهجٍ مرتبٍ ومنظمٍ وقابلٍ للقياس لعمليات تطوير وتشغيل وصيانة البرمجيات، أي تطبيق الهندسة على البرمجيات.

تهتم هندسة البرمجيات عموماً بتحليل البرمجيات وتصميمها وبناءها وتَحققها وإدارتها بشكل فعال.
وللقيام بتطبيق هندسة البرمجيات على وجه صحيح، يجب تعريف الإجرائية البرمجية software process، أي المنهج المتّبع لهندسة البرمجيات من اجراءات ، وعمليات ،ونشاطات ، ومهام ،واحداث ، وطرق ، وادوات، ونماذج ومبادئ هندسية في كل المنتجات البرمجية.

1-3- مشاكل فشل البرمجيات التي اوضحتها بعض الدراسات السابقة وهي:

The software which failed called runaway software result into software “crisis” , statistics show that only 2% from developed projects only used as they delivered, 3% of projects only used after modification,47% never used only delivered , 19% rejected or rework and 29% was not delivered.



1-4- معالجات البرمجيات Software Processes

- **معالجة البرمجيات :** هى مجموعة من الاجراءات (المعالجات) المترابطة والمتماسكة المطلوبة لتطوير وإنتاج النظم البرمجية.
- **والأنشطة العامة لتطوير منتج برمجي هى:** توصيف المتطلبات، التصميم، التنفيذ، الاختبار، التحقيق، الصيانة، ارتقاء النظم البرمجية.
- وتمثل هذه الأنشطة فى نماذج العمليات البرمجيات.
- مبدأ العمل في تطوير النظم البرمجية يعتمد على الاستخدام الامثل لمبدأ الطبقات والتي هي مناطق العمل التي ترتكز عليها تقنية تطوير النظم البرمجية وهي:

1. الادوات Tools

-هي مكونات مادية - ومكونات برمجية

2. الطرق Methods

-هي مجموعة الطرق والخوارزميات المختلفة ، لتحليل وتصميم النظام او البرامج ، وتمثيل انسياب البيانات .. وغيرها وتطبيق الادوات والطرق (النماذج- التوثيق- البيانات ،التقارير والاشكال) ومراقبتها وضمان جودة المنتج النهائي.

3. المعالجات (الإجرائيات) Processes

المعالجات تعتبر الاساس لمجموع الخطوات للازمة لتطوير وتسليم برمجيات خالية من العيوب والاخطاء ، موثوقة ، قابلة للاستخدام ، قابلة للصيانة ، تحقق متطلبات الزبون ، تسلم في الزمن المحدد ، وذات جودة عالية هو الاساس لهندسة المشروع البرمجي.

4. التركيز على الجودة A Quality Focus

لابد من التركيز على جودة المنتج البرمجي من اول نقطة البدء ويستمر هذا التركيز والحرص في كل الخطوات والطرق والقواعد المستخدمة بنية ضمان جودة المنتج البرمجي. وحيث تعتبر تحليل النظم المفتاح الاساسي لنجاح العمل في الأنظمة البرمجية وتحديدا ان تركيز التحليل ينصب على الاجرائيات والطرق والادوات.

1-5- تعريف معالجة البرمجيات Definition of Software Proce

تعريف معالجة البرمجيات بانها: الارضية للنشاطات والافعال والمهام تلك التي تكون مطلوبة لبناء برمجيات عالية الجودة.

معالجة البرمجيات Software Process : تعرف الطريقة المتبعة لتكون البرمجية مهندسة.

ملاحظة : معالجة البرمجيات لا تساوي هندسة البرمجيات والتي تتضمن ايضا نماذج وتقنيات وطرق ومبادئ وادوات اتوماتيكية ومواصفات وقياسات يتم تنفيذها على المنتج البرمجي اثناء تطويره.

1-5-2- تعريف المعالجة A process defined:

بانها مجموعة من الاعمال والنشاطات ، والافعال ، والمهام التي تؤدي عند انشاء اي منتج برمجي.

يمكن وصف عملية البرمجيات كما هو مبين في الشكل أدناه. حيث يوضح ارضية معالجة عامة **Common Process Framework** وذلك بتعريف عدد صغير من نشاطات الارضية التي يمكن تطبيقها على جميع المشاريع البرمجية بغض النظر عن حجمها أو تعقيدها، وعدد من مجموعات الافعال والمهام حيث كل منها عبارة عن مجموعة من: مهام عمل هندسة البرمجيات، مواصفات المشروع، مخططات وتصاميم ،عمل برمجية ونواتج عملية، ونقاط ضمان الجودة(التي تمكّن من تكييف نشاطات الارضية مع خصائص المشروع البرمجي ومتطلبات فريق العمل في المشروع). وأخيراً تغلف بنشاطات المظلة التي تضاف الى ارضية معالجة البرمجيات. إن نشاطات المظلة هي احداث مستقلة تجرى طوال عملية البرمجيات.

لكل ما سبق نجد ان معالجة البرمجيات Software Process تحتوي على ارضية المعالجات ، والتي تعتبر اللبنة الاساسية التي ينبني عليها كثير من المعالجات اوهي الطريق المتبع كي تنفذ هندسة البرمجيات على مراحل التطوير، وهي تحتوى على ارضية المعالجة (Process Framework) ، والتي بدورها تحتوي على ارضية النشاطات العامة والنشاطات الموسعة (المظلة):

ارضية النشاطات العامة لهندسة البرمجيات A Generic Framework For S/E

تتضمن خمسة نشاطات Encompasses Five Activities هي:

1-**التواصل Communication**
2- **التخطيط Planning**
3- **النمذجة Modeling**
4- **البناء Construction**
5- **الانتشار Deployment.**

شرح استخدامات النشاطات العامة لهندسة البرمجيات كما يلي:

1- الاتصال Communication

يستخدم التواصل بين الزبون والمهتمين بالمنتج البرمجي (مهندسو البرمجيات ، المحللون للنظام، المصممون، البناؤون ، المختبرون، الزبائن ، المستخدمون الحقيقيون للنظام) ، وتتضمن جمع المتطلبات من كل المتواصلين.

2- التخطيط Planning

تؤسس خطة عمل لهندسة المنتج :المهام التقنية ، المصادر ، عمل المنتج ، وجدول تنفيذ العمل.

3- النمذجة (تحليل ، تصميم) (Modeling (Analyze, Design)
يتضمن انشاء واستخدام نماذج لتحسين فهم المتطلبات والتصميم للنظام .

4- البناء(تطبيق وفحص) Construction (Implementation and Testing)
تجمع بين توليد الشفرة البرمجية وفحص الاخطاء الغير مكتشفة (اي تنفيذ النظام)

5- النشر Deployment
تستخدم لتسليم المنتج البرمجي للزبون لتقييمه واعطاء ملاحظاته كتغذية راجعة.

5-5-1- النشاطات الموسعة (المظلة) Umbrella Activities
ارضية نشاطات هندسة البرمجيات: تحتوي على الخمسة النشاطات العامة وتكون مكملة بواسطة العديد من عمليات النشاطات الموسعة (نشاطات المظلة) .وبشكل عام النشاطات الموسعة تطبق عند تنفيذ المشروع البرمجي لإدارة وتوجيه فريق المشروع ومراقبة تطويره مع الخطة ومراقبة الجودة والتغيير، والمخاطر .وهذه النشاطات هي كما يلي:

6- ادارة مشاريع البرمجيات -Software project management
تسمح لفريق البرمجية تقييم الانجاز ومقارنته بخطة المشروع واخذ التدابير اللازمة للتوافق مع الجدول.

7- ادارة المراجعات التقنية الرسمية -Formal technical reviews
تساعد مهندس البرمجيات في مضاعفة الجهود لكشف واجتثاث الاخطاء من المنتج قبل ان تتكاثر وتنتقل الى النشاطات التالية.

8- ادارة المخاطر -Risk management
تسهل عملية تقييم المخاطر التي ربما تؤثر على مخرجات المشروع او على جودة المنتج.

9- ادارة الجودة للبرمجيات -Software quality assurance
تعرف وتحدد النشاطات المطلوبة لضمان جودة البرمجية

10- ادارة المعايير(المقاييس) -Measurement Management
تعريف وتجميع المعالجة والمشروع وانتاج المقاييس التي تساعد الفريق في تسليم برنامج يحقق متطلبات الزبائن ، ويكون استخدامها بالتزامن مع ارضية النشاطات العامة والموسعة(المضلة).

11-ادارة اعداد البرمجيات -Software configuration management
تقوم بادراة التغييرات واثارها الناتجة اثناء تطوير البرمجية

- **12-ادارة اعادة الاستخدام – Reusability management**

- defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.

- **13- ادارة التحضيرات ولإنتاج – Work product preparation and production**

- encompasses the activities required to create work products such as models, documents, logs, forms, and lists.
-

الفصل الثاني

نماذج هندسة البرمجيات

Software Process Models

2-1- نماذج معالجة البرمجيات Software Process Models

لكي يتمكن مهندس البرمجيات أو فريق المهندسين من تحليل مسائل حقيقية في بيئة التطوير، عليهم استخدام استراتيجية هندسة البرمجيات والتي ترتكز على الإجرائية البرمجية او (معالجة البرمجيات Software Process) والتي تعتمد على(العمليات ، والنشاطات، والمهام ، والاحداث، الطرق والحرص على الجودة المشروحة سابقا، ثم يضيفوا الى ماسبق استخدام القوانين والمبادئ الهندسية (Principles) العامة والخاصة ثم يختاروا نموذج من نماذج معالجة البرمجيات اعتماداً على طبيعة المشروع وتطبيقاته ، هذا النموذج يعتبر كقالب لمواصفات وخطوات وتسلسل وطرق متتابعة او مكررة او متزايدة) والذي سيتم العمل بها واتباعها في كل معالجة من معالجات البرمجية ، بل ويتم تطبيقها على مراحل التطوير وكالاتي:

2-2- نشاطات اساسية لكل معالجات البرمجيات

Fundamental Activities Are Common To All Software Processes

1-توصيف النظام: يتم هنا التوصيف الوظيفي للنظام، ومعرفة حدوده والعمليات التي سيقوم بها يجب معرفتها وتحديدها وتوصيفها.

2-تصميم النظام وتنفيذه : تصميم معمارية واجزاء النظام كاملة ثم وبناءه بكتابة الكود البرمجي وفقا للتصميمات المعدة ليقابل التوصيفات المقدمة اولا.

3- تحقيق النظام : فحص واختبار النظام حتى يضمن انه يعمل وفقا لمتطلبات الزبون ويحقق رغباته.

4-ارتقاء النظام: النظام يجب ان يكون مرنا ويستجيب لمقتضيات التغيير المحتاجة من الزبون ويواجه التعديلات لتنفيذ وظائف او متطلبات جديدة.

2-3- انواع نماذج معالجة البرمجيات

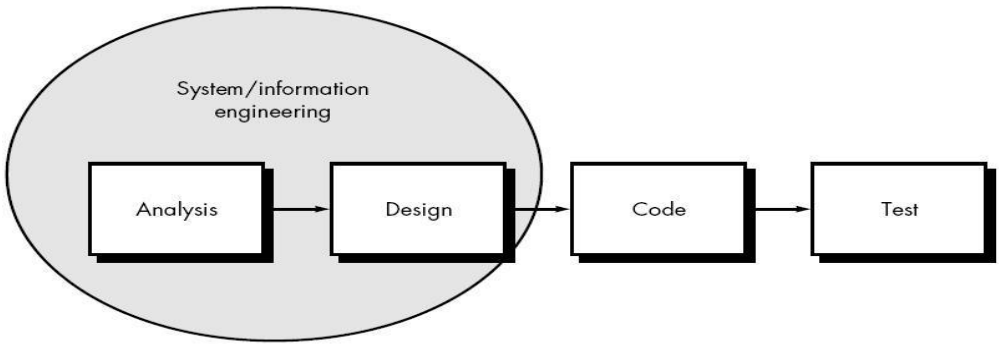
يتم اختيار نموذج لإجراء معالجة البرمجيات بناءً على طبيعة المشروع ومواصفاته وتطبيقاته ، الطرق والأدوات المستخدمة، والتحكم والأداء المطلوبين.

سوف نناقش في الفقرات القادمة نماذجاً مختلفة للعمليات البرمجية المتبعة في هندسة البرمجيات حيث يمثل كل نموذج منها محاولة لإضفاء ترتيب وتنظيم على نشاط عشوائي (غير منتظم) وفق طريقة معينة.

اي ان النموذج عبارة عن وصف او طريقة او تمثيل محدد لتبسيط المعالجات البرمجية وكيفية اجرائها عند تطوير البرمجيات.

ولذا عندما ننظر الى النماذج من عدة اتجاهات نجد انها تنقسم الى ثلاثة اقسام رئيسية هي:

FIGURE
The linear sequential model



أ - النموذج التتابعي الخطي:(Linear sequential)

ويعتمد طريقة «دورة الحياة التقليدية». وهو منهج تتابعي خطي منتظم لتطوير البرمجيات، يبدأ من المرحلة الاولى مرحلة تعريف النظام ثم يتقدم تباعا إلى التحليل ، فالتصميم ، فالبناء ، ثم الاختبار، فالصيانة. الشكل اعلاه

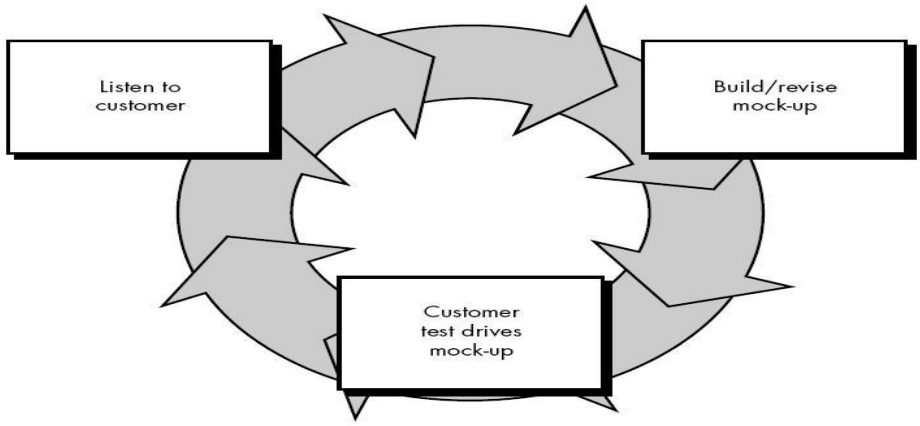
ب - النماذج التكرارية Iterative Models

لا تستخدم هذه النماذج الطريقة التسلسلية في تطوير البرمجيات بل انها تستخدم طريقة التكرارات ، بحيث في كل تكرار(دورة) سيتم انجاز جميع مراحل تطوير البرمجيات (توصيف ، تصميم ، بناء ، اختبار) ، غالباً ما يعرِّف الزبون مجموعة من الأهداف العامة للبرمجيات المطلوبة، ولا يحدِّد بالتفصيل متطلبات كل من الدخل والمعالجة أو الخرج. لذلك يقدِّم نموذج «النمذجة الأولية» prototyping paradigm الطريقةَ الفضلى في هذه الحالات. تبدأ النمذجة الأولية بجمع المتطلبات، لذلك يجتمع المطوّر والزبون لتعريف الأغراض الإجمالية للبرمجيات، وتحديد أية متطلبات معروفة، ويوضع عندئذٍ "تصميم سريع". ويقود التصميم السريع إلى بناء نموذج أولي prototype. يُعيد الزبون تقييمَ النموذج الأولي، ويُستخدم ذلك لتحديد المتطلبات للبرمجيات اللازم تطويرها. ويحدث التكرار من خلال ضبط النموذج الأولي لتحقيق متطلبات الزبون.

ج ـ نماذج التطوير الارتقائي Evolutionary Development Models

هناك اعتراف متزايد إن البرمجيات، مثل جميع الأنظمة المعقدة، تتطور على مر الزمن إذ تتغير غالبا متطلبات الأعمال والمنتج مع تقدم عملية تطوير هذا المنتج، وهذا ما يجعل المسار المستقيم باتجاه إنتاجه غير واقع، يضاف إلى ذلك إن المواعيد المقيدة لطرح المنتج في السوق تجعل إكمال منتج برمجي شامل شيئا مستحيلا، ولكن عندما نرغب بتقديم نسخة محدودة لمواجهة المنافسة أو ضغط العمل

FIGURE
The prototyping paradigm



نجد هذا النموذج جيدا شرط أن تكون نواة المنتج أو متطلبات النظام مفهومة جدا، ولو لم تعرف بعد توسيع المنتج أو النظام، يحتاج مهندسو البرمجيات في هذه الحالات وفي حالات أخرى مشابهة إلى نموذج العملية البرمجية تصمم بوضوح (بالتفصيل) لاستيعاب منتج يتطور مع الزمن.

لقد صمم النموذج ألتتابعي الخطي لحالات التطوير المباشر (خط مستقيم) وبمعنى آخر، تفترض هذه الطريقة الشلالية انه سيتم تسليم كامل النظام بعد اكتمال هذا التتابع الخطي، ومن جهة أخرى فقد صمم النموذج الأولي لمساعدة الزبون أو (المطور) على فهم المتطلبات، ولم يصمم عموما لتسليم نظام نهائي.

فلَم تُلاحَظ الطبيعة التطورية للبرمجيات في كلا هذين النموذجين التقليديين لهندسة البرمجيات.

النماذج التطورية (evolutionary models) تكرارية، وتوصَّف بطريقة تمكِّن مهندس البرمجيات من تطوير نسخ أكثر تعقيداً من البرمجيات، وسنذكر فيما بعداثنين من هذه النماذج.

بعد ان تم التعرف على ثلاثة انواع من النماذج سوف تتعمق الفكرة للقارئ من خلال اعطاء امثلة توضيحية عن كل نوع من النماذج وسنتطرق في شرحنا لكل نموذج من النماذج الثلاثة بطريقة تساعد على التحكم والتنسيق في المشاريع البرمجية الحقيقية وهي كالتالي:

2-2- امثلة للنماذج التتابعية الخطية (Linear sequential)

2-2-1-النموذج الشلال: Waterfall

هو نموذج تتابعي خطي يقترح منهجاً تتابعياً منتظماً ، لتطوير البرمجيات. يلتزم باتباع "دورة الحياة التقليدية" المتعاقبة كالشلال" والذي يطلق عليه" نموذج الشلال" (waterfall model)

تعتبر طريقة نموذج الشلال من الطرق التقليدية القديمة ، ورغم ذلك فهي طريقة اساسية معتمدة في تطوير البرمجيات ومعالجة نشاطات التطوير التي هي:

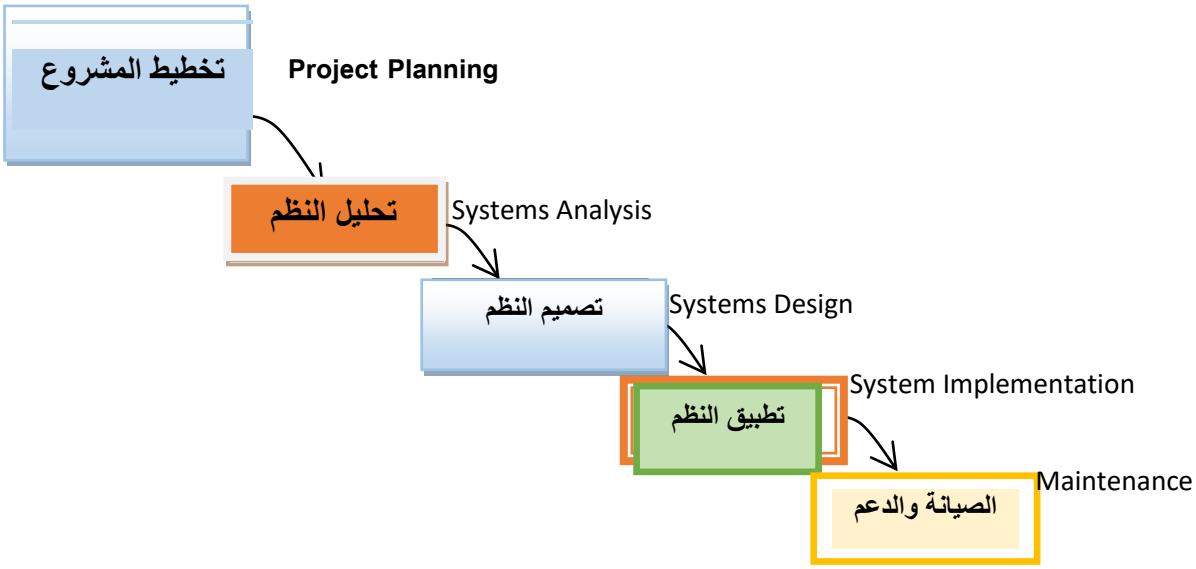
1- تعريف وتوصيف متطلبات النظام

2- التصميم تصميم معمارية النظام واجزائه

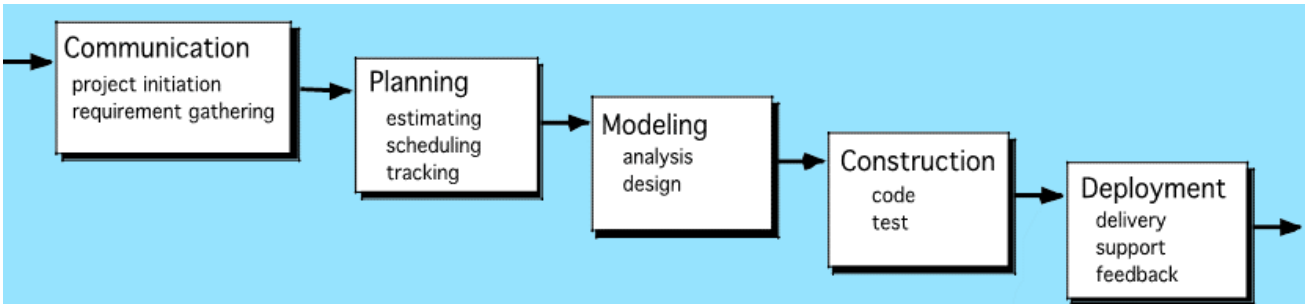
3- ثم التطبيق (كتابة البرامج واختبارها)

4- ثم الصيانة والدعم

بحيث تعالج كل مرحلة من مراحل التطوير باستقلالية تامة عن المراحل اللاحقة ، ولايتم الانتقال من اي مرحلة الى التي تليها الا بعد استكمال كل اعمال هذه المرحلة . ولا يتم الرجوع الى الخطوة السابقة اطلاقا



الشكل(6.1): النموذج التدفقي (الشلال) Water Fall لدورة تطوير حياة



اولا: يكون هذا النموذج ناجحا عندما تكون كل المتطلبات ثابتة محددة واضحة ومفهومة تماما .

ثانيا: من المشاكل او العيوب التي تظهر أحيانا عند تطبيق النموذج ألتتابعي وهي :

العيـــــــــوب:

1- تكاليف التعديلات المتأخرة :

نادراً ما تتبع المشاريع الحقيقية التقدم ألتتابعي الذي يقترحه النموذج الشلالي، ومع إن النموذج يمكن أن يستوعب التكرار، إلا انه يفعل ذلك بأسلوب غير مباشر، ونتيجة لذلك، قد تسبب التعديلات ارتباكا (فوضى) مع تقدم فريق المشروع.

2- صعوبة طرح وتحديد وتوصيف جميع متطلبات الزبون دفعة واحدة:

يصعب غالبا على الزبون طرح جميع متطلباته بوضوح، والنموذج الخطي يحتاج إلى ذلك، ولهذا تبرز صعوبات في استيعاب عدم التحديد الطبيعي للمتطلبات الذي يتم في العديد من المشاريع.

3- يجب أن يتحلى الزبون بالصبر:

إذ لن تتوفر نسخة عاملة من البرمجية او (النظام) حتى وقت متأخر من الجدول الزمني للمشروع، وقد يكون الخطأ المتأخر كارثيا إذا لم يكتشف إلا عند مراجعة البرنامج العامل (المنتج النهائي).

4- غالبا ما يتأخر المطورون لأسباب غير ضرورية

إن الطبيعة الخطية لدورة الحياة التقليدية تقود إلى "حالات انتظار"(blocking states) يضطر فيها بعض أعضاء فريق المشروع انتظار أشخاص آخرين من الفريق لإنهاء مهام مترابطة (غير مستقلة)، وفي الواقع قد يتجاوز وقت الانتظار هذا الزمن المصروف على اعمال منتجة هناك مثل كان تحصل حالات انتظار بكميات اكبر في بداية ونهاية عملية البرمجة التتابعية الخطية.

ومع إن كل هذه المشاكل هي مشاكل حقيقية، إلا إن لنموذج دورة الحياة التقليدية مكانا محددا وهاما في عمل هندسة البرمجيات، فهو يزودنا بقالب توضع فيه طرق للتحليل والتصميم والبناء والاختبار والصيانة ولا تزال دورة الحياة التقليدية لهندسة البرمجيات هي نموذج عملية البرمجة الأكثر استخداماً. وبالرغم من وجود نقاط ضعف فيها، إلا أنها أفضل بكثير من تطوير البرمجيات وفق منهج المصادفة.

عيوبه:

1- **تكاليف التعديلات المتأخرة**
2- **صعوبة طرح وتحديد وتوصيف جميع متطلبات الزبون دفعة واحدة**
3- **يجب أن يتحلى الزبون بالصبر**
4- **غالبا ما يتأخر المطورون لأسباب غير ضرورية**

مميزاته:

1- **نموذج الشلال يزودنا بقالب لطرق التحليل والتصميم والبناء والاختبار والصيانة**
2- **يعتبر نموذج مناسبا يحاكي عملية البرمجة الأكثر استخداما**
3- **تستخدم خطواته في كثير من النماذج الأخرى (التزايدي ، والنموذج الاولي ، و نموذج المعالجة الموحدة..)**
4- **يكون مناسبا استخدم هذا النموذج مع المشاريع الحكومية الكبيرة.**

2-2-2-نموذج التطوير السريع Rapid Application Development (RAD)

التطوير السريع للبرنامج هو نموذج تتابعي خطي لعملية تطوير البرمجيات يهتم بدورة تطوير قصيرة جدا (من 60 إلى 120 يوما) ، النموذج RAD هو تكييف "سريع جدا" للنموذج ألتتابعي الخطي، حيث يجري تطوير سريع باستخدام أسلوب بناء يعتمد على المكونات إذا فهمت المتطلبات جيدا وحصرت أفق المشروع( ) ، فان عملية البرمجة RAD تسمح لفريق التطوير بإيجاد **"نظام كامل يعمل تماماً" (fully functional system)** ،يتضمن الأسلوب RAD، الذي تم استخدامه استخداما أوليا في تطبيقات أنظمة المعلومات، المراحل التالية:

نمذجة الأعمال (Business Modeling) : نماذج تدفق المعلومات بين وظائف الأعمال بطريقة تجيب عن الأسئلة التالية: ما المعلومات التي تقود (تسير) عملية برمجة الأعمال؟ ما المعلومات التي يجري توليدها؟ من الذي يولدها؟ أين تذهب المعلومات؟ من يعالجها؟

نمذجة البيانات (Data Modeling): تلخص أولا تدفق المعلومات، الذي عرف انه جزء من مرحلة نمذجة الأعمال وتدفقه، في مجموعة من أهداف البيانات اللازمة لدعم الأعمال ثم نحدد المميزات (تسمى سمات، attributes) لكل هدف ويتم تعريف العلاقات بين هذه الأهداف.

نمذجة عملية البرمجة (Process Modeling): تحول أهداف البيانات، التي تم تعريفها في مرحلة نمذجة البيانات لتحقيق تدفق المعلومات الضروري، اللازمة بدورها لتحقيق وظيفة الأعمال، ثم نعمل على توليد سمات معالجة لكل عملية من عمليات إضافة أو تعديل أو حذف أو استرجاع أهداف البيانات.

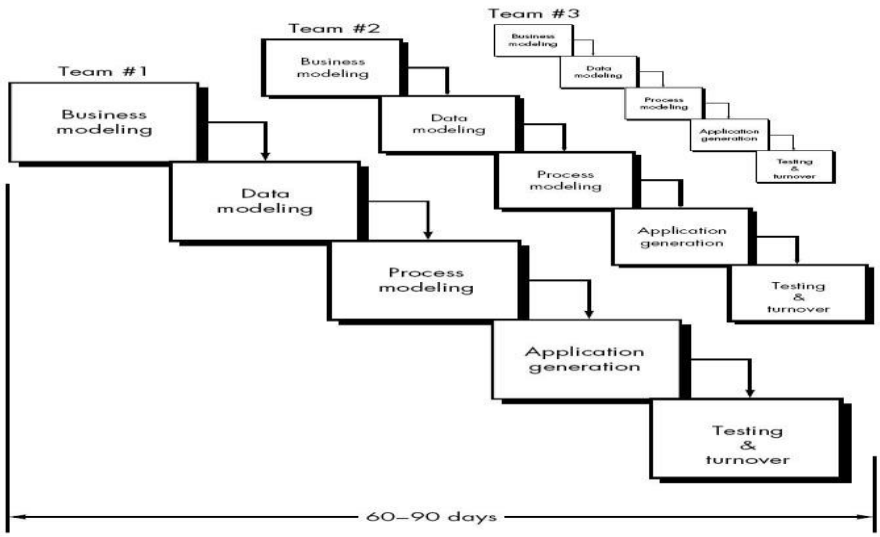
توليد التطبيق(Application Generation) : يفترض التطوير السريع RAD استخدم تكنولوجيات الجيل الرابع (GT) (Fourth Generation Technology)) . فبدلا من إيجاد برمجيات باستخدام لغات برمجة تقليدية من الجيل الثالث، تعمل عملية التطوير السريع للبرنامج RAD على إعادة استخدام مكونات البرنامج الموجودة (عندما يكون ممكنا)أو إنشاء مكونات قابلة لإعادة الاستخدام (عند الضرورة) وتستخدم الأدوات المؤتمتة في كل الأحوال لتسهيل بناء البرنامج.

الاختبار(Testing): لما كان التطوير السريع للبرنامج RAD يشدد على إعادة الاستخدام، سيكون قد سبق وتم اختبار العديد من مكونات البرنامج، وهذا ما يقلل من زمن الاختبار الكلي ومع ذلك يجب اختبار المكونات الجديدة وتجربة كل الواجهات تجريبا كاملا.

متطلبات نموذج RAD

- يتطلب نفقات وموارد بشرية كافية لإنشاء العدد المطلوب من الفرق في حالة المشاريع الكبيرة.
- يتطلب مطورين وزبائن ملتزمين بالنشاطات السريعة والمتلاحقة الضرورية لإتمام النظام في زمن مختصر جدا
- يمكن تناول كل وظيفة رئيسة بواسطة فريق RAD مستقل، ثم تتكامل لتشكل كيانا واحداً.

FIGURE
The RAD model



مميزات نموذج RAD

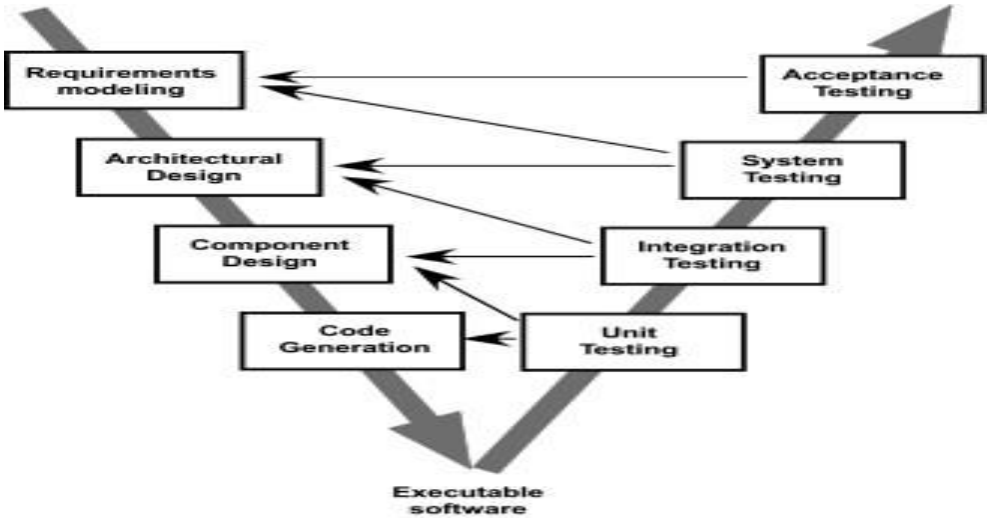
- سرعة الانجاز نظرا لتقسيم العمل

عيوب نموذج RAD

- RAD غير مناسب عندما تكون المخاطر التقنية عالية.
- النماذج لها بعض المساوئ وليست كل أنماط التطبيقات مناسبة لإعادة الاستعمال.
- إن التقييد الزمني المفروض على مشروع RAD

2-2-3- نموذج التحقق V-Model

اولا: الفريق يتحرك في الجانب الايسر لكي يراجع مشكلة المتطلبات ، وعندما يكتب الكود البرمجي لهذه المشكلة ، يعود الفريق الى الاعلى من الجانب الايسر من حرف V ، ويؤدي سلسلة من الاختبارات الذي يحقق كل النماذج التي تكونت في الجانب الايسر.
يكون الفريق البرمجي يتحرك في الجانب الايسر، تكون اساس مشكلة المتطلبات قد نقحت وزودت وعرضت على كثير من التفاصيل والحلول التقنية..
وعندما يولد الكود البرمجي مرة ثانية يعود الفريق الى الاعلى يمينا من V اساسا محدثا كثير من الاختبارات (لعملية فحص الجودة) والذي يمكنهم من تحقيق كل النماذج المكونة عندما كان نازلا بالجانب الايسر



-في الحقيقة لا يوجد فرق بين نموذج V والطريقة التقليدية.
النموذج V يوفر طريقة واضحة لكيفية الفحص والتحقيق او التصديق والتي تطبق على العمل الهندسي المسبق.

مميزات هذا النموذج

- لا يتم تعطيل كل اعمال المراحل التالية الى ان يتم انجاز المرحلة الحالية.
- معظم فريق العمل يكون مشغولا وفاعلا في انجاز مهما التطوير اولا بأول.
- عملية الفحص والتأكيد اولا بأول تقوي رصانة المنتج البرمجي وتوضح مكوناته اولا بأول.
- يستخدم هذا النموذج عندما تكون المتطلبات غامضة ولن توضح الا بعد سلسة من التجارب.
- جاء هذا النموذج للتغلب على مشاكل نموذج الشلال .

عيوب هذا النموذج

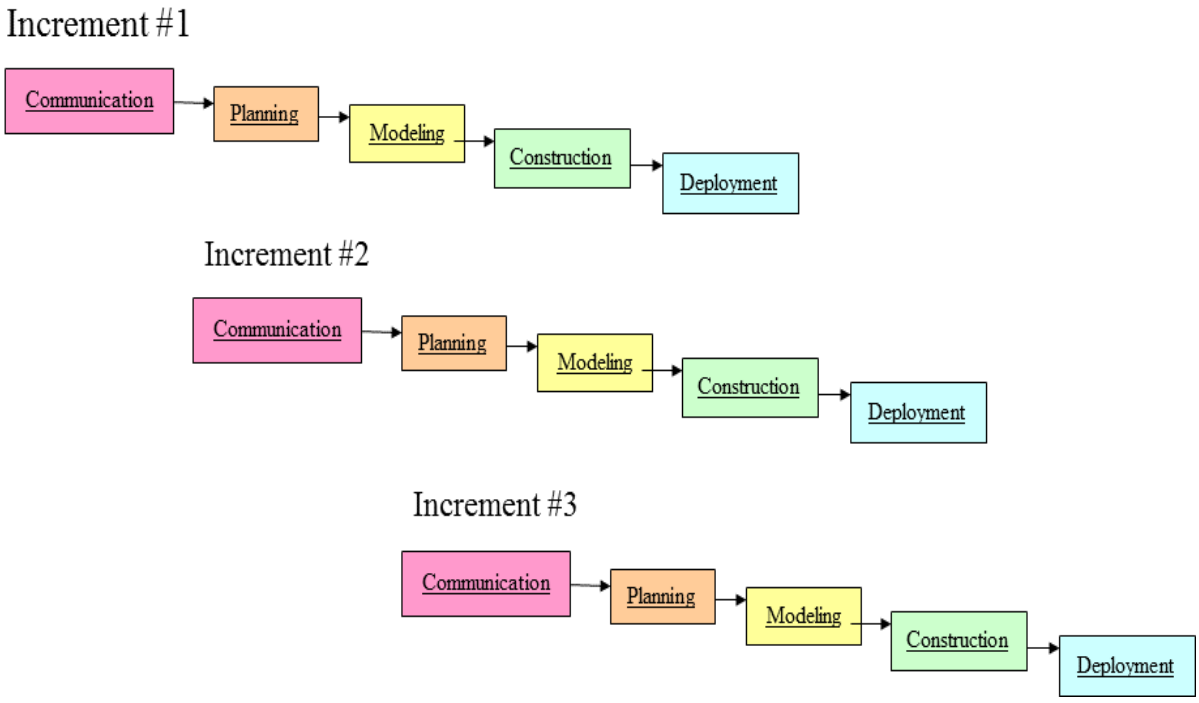
- يحتاج الى فريق مؤهل تماما ومتعاون ونشيط جدا
- في بعض الاحيان يكون مكلفا ويؤدي الى فشل المنتج.

2-3- النماذج التكرارية Iterative Models

2-3-1- النموذج ألتزايدي : (incremental model)

عندما المتطلبات الاولية تكون معروفة تماما ، ولكن بقية خطوات التطوير المتعاقبة غامضة.

يجمع النموذج ألتزايدي عناصر من النموذج ألتتابعي الخطي (مطبقة بصورة متكررة) مع الفلسفة التكرارية. وعليه يقوم النموذج ألتزايدي (كما في الشكل اللاحق) بتطبيق تتابعات خطية بأسلوب متعاقب مع تقدم زمن الإنتاج (الجداول الزمنية للإنتاج) وينتج كل تتابع خطي تزايدا من البرنامج يمكن تسليمه للزبون .



فمثلا قد تقدم برمجيات معالجة النصوص المطورة باستخدام النموذج ألتزايدي إدارة أساسية للملفات والتحرير ووظائف إنتاج الوثائق في التزايد الأول، وتقدم إمكانيات أكثر تطورا (تعقيدا وتقدما) كتحرير وإنتاج وثائق في التزايد الثاني، وتقدم التصحيح الإملائي والقواعد في التزايد الثالث، وتقدم إمكانات متقدمة لضبط تنسيق الصفحة (page layout) في التزايد الرابع، ومن الجدير بالذكر انه يمكن استخدام نموذج النمذجة الأولية في سير عملية البرمجة في أي تزايد.

عند استخدام التزايد الأول: فانه يتم تحديد المتطلبات الأساسية، فيخضع لمراجعة تفصيلية، ويحفز نتيجة للاستخدام أو للتقييم تطوير خطة التزايد التالي، وتحدد الخطة تعديلات التزايد الاول لتلبية حاجات الزبون بشكل أفضل وتقديم مزايا ووظائف إضافية، وتتكرر هذه العملية بعد تسليم كل تزايد حتى يتم إنتاج كامل النظام.

إن النموذج ألتزايدي هو تكراري بطبيعته مثل النمذجة الأولية وجميع الأساليب التطورية، ولكنه بخلاف النمذجة الأولية يحرص على تقدم منتج عامل في كل تزايد، تكون التزايدات الأولى "نسخا منسوخة" عن المنتج النهائي، ولكنها تستطيع تقديم إمكانيات تفيد المستخدم وتقدم أيضا منصة عمل (Platform) لتقيمها.

مميزات النموذج التزايدي

- لا يشترط تحديد المتطلبات كاملة في البداية.
- لايحتاج الى كثير من العاملين فيمكن انجاز التزايدات الأولية بعدد قليل من العاملين.
- تقليل المخاطر وضمان عدم انهيار المشروع كليا.
- مشاركة الزبون في التخطيط والتقييم مع كل جزء يقوي رصانة النظام مبكرا.

عيوب النموذج التزايدي

- صعوبة تحديد القيود الصارمة عند التعاقد
- قد تحدث مشاكل غير متوقعة عند دمج النسخ الجديدة مع القديمة.

2-4- امثلة: النماذج التطورية Evolutionary Models

لقد صُمِّم النموذج التتابعي الخطي لحالات التطوير المباشر (خط مستقيم) ، وتفترض هذه الطريقة التتابعية أن النظام كله سيسلَّم بعد اكتمال هذا التتابع الخطي. ومن جهة أخرى، فقد صُمِّم نموذج التكراري لمساعدة الزبون (أو المطوّر) على فهم المتطلبات، ولم يصمم عموماً لتسليم نظام نهائي. لم تلاحظ الطبيعة التطورية للبرمجيات في كلا النموذجين السابقين لهندسة البرمجيات فتم تصميمها في النموذج الارتقائي التطوري والذي نبدأه بالنموذج الارتقائي الاولي:-

2-4-1-النموذج الارتقائي "الأولي" Evolutionary Model: Prototyping

يستخدم النموذج الأولي كآلية لتحديد متطلبات البرنامج وبناء نموذج أولي عندما:

1- لا يعرف الزبون مجموعة من الأهداف العامة للبرامجية، ولا يحدد بالتفصيل كل متطلبات الادخال أو المعالجة أو المخرجات

2-المطور غير متأكد من فعالية الخوارزمية المقترحة للبرمجية، أو من تكيف نظام التشغيل، أو الشكل الذي يجب أن يأخذه تفاعل الإنسان مع الآلة.

لذلك فان نموذج النمذجة الأولية (Prototyping Paradigm) يقدم الطريقة الفضلى في هذه الحالات وحالات كثيرة غيرها.

تبدأ استخدام النمذجة الأولية (الشكل) بجمع المتطلبات لذلك يجتمع المطور والزبون لتعريف الأهداف الإجمالية للبرمجية، وتحديد أية متطلبات معروفة، وتحديد عناوين المجالات التي تتطلب تعريفات أكثر ويوضع عندئذ "تصميم سريع" يركز التصميم السريع على تمثيل نواح محددة من البرمجية، وخاصة تلك التي ستكون مرئية للزبون أو المستخدم (كطرق الإدخال وصيغ الإخراج).

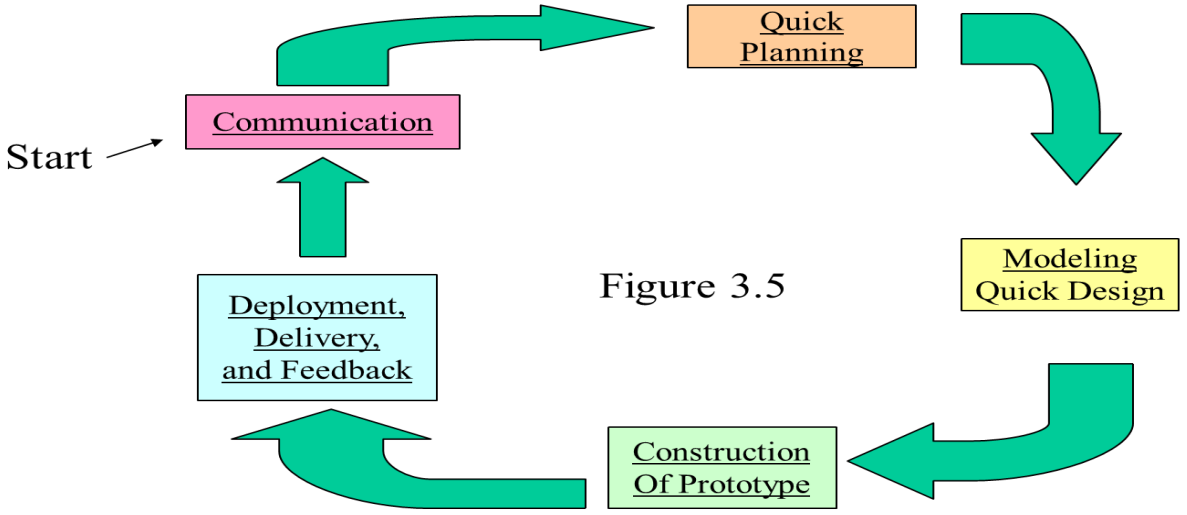


Figure 3.5

يقود التصميم السريع إلى نموذج أولي (Prototype) يسلم للزبون لتقييمه ، يعيد الزبون أو المستخدم تقييم النموذج الأولي ويستخدم ذلك التقييم لتوصيف متطلبات البرمجية التي يجب تطويرها ويحدث تكرار من خلال ضبط النموذج الأولي لتحقيق متطلبات الزبون، وهذا يمكّن المطور في الوقت ذاته من إيجاد فهم أفضل للحاجات الواجب تلبيتها.

هنالك طريقة فعالة ومتبعة كثيرا في تطوير المشروع البرمجي بحيث يتم انشاء نماذج أولية للمشروع وتتطور هذه النماذج لتصبح نهاية الامر النظام البرمجي الحقيقي الكامل. وتعتبر تلك النماذج للنظام عبارة النظام في طور النضج إلى ان يصبح متكاملا .
تتخذ القرارات تدريجيا في التغيرات التي تطرأ على النماذج الأولية مما يساهم في الوصول إلى القرارات النهائية بشكل صحيح ونهائي ويمكن اعتبار أن النماذج الشبه ناضجة او كاملة بمثابة النسخ الاولى عن النظام البرمجي (في بعض الاحيان) ويمكن استخدامها وبحيث تعتبر بمثابة اختبار حقيقي للنظام وعلى مدى نضج النظام . هنالك نماذج متعددة من Prototyping models

مشاكل النموذج الاولي:

1. **يرى الزبون في النموذج الأولي ما يبدو انه نسخة صحيحة (عاملة) من البرنامج وهو غير مدرك إن هذا النموذج الأولي قد حوفظ عليه مترابطا ترابطا واهيا جدا**، ولا يدرك انه بسبب السرعة في انجازه لم تؤخذ بالحسبان الجودة الكلية للبرنامج أو لقابلية صيانته(Maintainability) على المدى الطويل. وعندما يتم إبلاغ الزبون بوجوب إعادة بناء المنتج بحيث يراعى تحقيق معايير عالية الجودة فانه يصرخ كالمجنون ويطلب إجراء "بعض الإصلاحات" لجعل النموذج الأولي منتجا يعمل جيدا، وغالبا ما تستجيب إدارة تطوير البرمجيات لهذا الطلب.
2. **غالبا ما يجري المطور بعض التجاوزات في الانجاز (Implementation) لجعل النموذج الأولي يعمل بسرعة.** فقد يستخدم نظام تشغيل أو لغة برمجة غير مناسبين، فقط لأنهما متوافران ومعروفان، وقد يعتمد خوارزمية غير كافية، فقط لإيضاح الإمكانيات. وقد يصبح المطور بعد مدة أكثر معرفة والماماً بهذه الخيارات وينسى كل أسباب كونها غير مناسبة، ويصبح الآن الخيار الأقل مثالية جزءاً متكاملاً من النظام.

بالرغم من إمكانية حصول مشاكل إلا إن استخدام النموذج الأولي يمكن أن يكون منهجا فعالا لهندسة البرمجيات والسر هو تحديد قواعد اللعبة منذ البداية، أي ، يجب أن يتفق المطور والزبون على إن النموذج الأولي يبنى لكي يستخدم كآلية لتعريف المتطلبات، ثم يهمل (على الأقل جزئيا) وتبنى البرمجيات الفعلية مع مراعاة الجودة والصيانة.

مميزات هذا النموذج

- امكانية الانتقال من مرحلة الى اخرى دون تخوف
- يصلح للمشاريع الصغيرة والمتوسطة الحجم

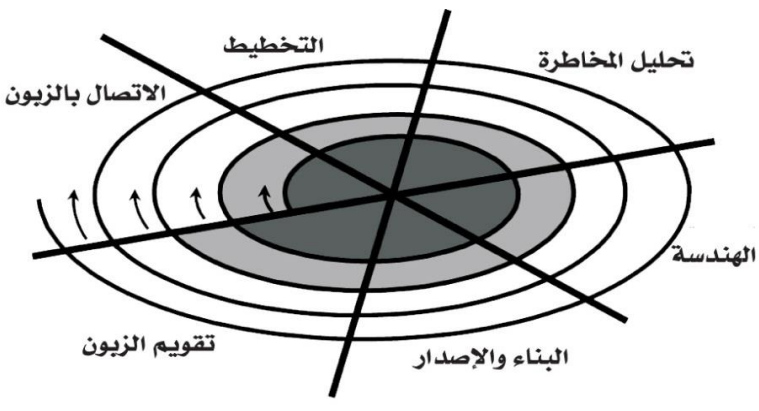
عيوب هذا النموذج

- يتطلب من المطورين مهارة عالية في مجال التطوير
- عدم وضوح الخطوات يؤثر على المعالجات

2-4-2- النموذج الحلزوني: The Spiral Model

النموذج الحلزوني (Spiral model) الذي اقترحه أولا Boehm هو نموذج تطوري لعملية البرمجة، ويقرن الطبيعة التكرارية للنمذجة الأولية بالنواحي النظامية والمحكومة للنموذج ألتتابعي الخطي، ويقدم إمكانية تطوير سريع لنسخ تزايدية من البرنامج، يتم تطوير البرمجية حسب النموذج الحلزوني في سلسلة من الإصدارات التزايدية، وقد يكون الإصدار ألتزايدي خلال التكرار الأولي نموذجا ورقياً أو نموذجاً أولياً، ويجري إنتاج نسخ أكثر اكتمالا من النظام الذي تمت هندسته خلال التكرارات التالية.
ينقسم النموذج الحلزوني إلى عدد من نشاطات الهيكل (framework activities) تسمى أيضا منطقة المهمة (task region) ، هناك عادة ما بين ثلاث إلى ست مناطق يمثل الشكل الاحق **نموذجا حلزونيا يحتوي على ست مناطق هي :**

- **الاتصال بالزبون** – المهام اللازمة للتواصل الفعال بين المطور والزبون .
- التخطيط – المهام اللازمة لتعريف الموارد، المسارات الزمنية (Timelines)، معلومات أخرى متعلقة بالمشروع.
- **تحليل المخاطرة** – المهام اللازمة لتقييم المخاطرة التقنية اولا بأول.
- **الهندسة** – المهام اللازمة لبناء تمثيل أو أكثر للتطبيق.
- **التشييد والإصدار (Construction & Release)** – المهام اللازمة لبناء واختبار وتثبيت وتقديم دعم للمستخدم (كالتوثيق والتدريب).
- **تقييم الزبون (Customer Evaluation)** – المهام اللازمة للحصول على التغذية الراجعة بالاعتماد على تقييم الزبون للإصدارات البرمجية التي انشات خلال مرحلة الهندسة وتم انجازها خلال مرحلة التثبيت.

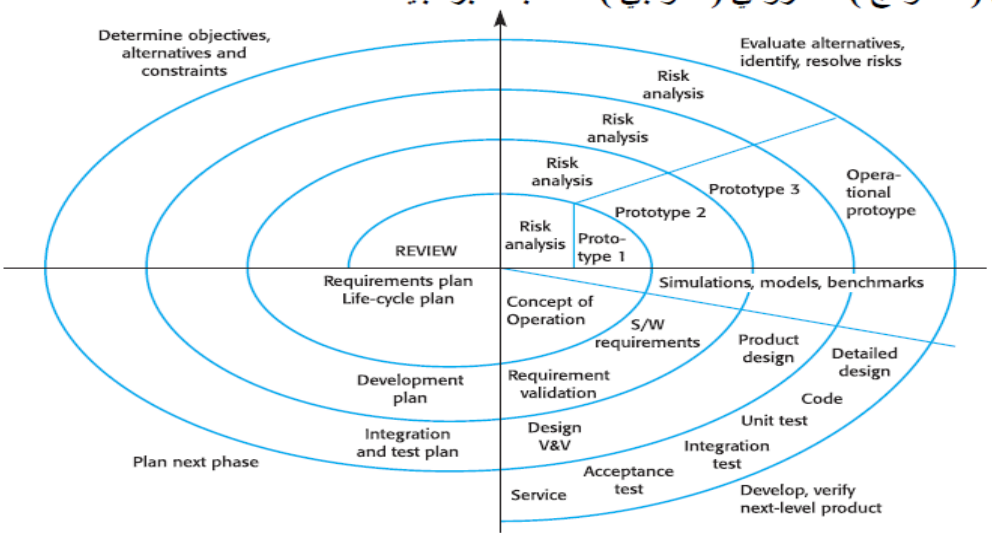


الشكل (٢) النموذج الحلزوني

كل منطقة من هذه المناطق الستة مأهولة بسلاسل من مهام العمل التي جرى تكييفها مع خصائص المشروع الذي يجري تنفيذه، يكون عدد مهام العمل الرسمية المتعلقة بها منخفضا في حالة المشاريع الصغيرة ، إما في المشاريع الكبيرة فتحتوي كل منطقة على مهام عمل أكثر، يجري تعريفها لتحقيق مستوى أعلى من الاداء.

Spiral model of the software process

التصميم (النموذج) الحلزوني (اللولبي) لمعالجة البرمجيات



تطبق في جميع الحالات نشاطات المظلة(مثل إدارة تشكيلة البرمجيات وضمان جودة البرمجيات)

حالما تبدأ عملية البرمجة التطورية، يتحرك فريق هندسة البرمجيات حول الحلزون باتجاه عقارب الساعة بدءاً من النواة ، وقد ينتج عن الدورة الأولى حول الحلزون تطوير مواصفات المنتج، وقد تستعمل الدورات التالية حول الحلزون لتطوير نموذج أولي، ثم شيئاً فشيئاً تعد نسخ من البرنامج أكثر تطورا ينتج عن كل مرور عبر منطقة التخطيط ضبط لخطة المشروع، ويجري ضبط الكلفة والجدول الزمني بالاعتماد على تعليقات الزبون إضافة إلى ذلك يضبط مدير المشروع عدد الدوران المخطط لها والمطلوبة لإنجاز المشروع.

خلافا للنماذج التقليدية لعملية البرمجة التي تنتهي عند تسليم البرنامج، يمكن تكييف النموذج الحلزوني لتطبيقه على امتداد كامل حياة البرنامج، يعرف الشكل أعلاه محور نقطة دخول المشروع (Point Project Entry) يمثل كل مكعب يوضع على هذا المحور نقطة البداية لمشروع جديد آخر.

يبدأ مشروع تطوير المفاهيم (Concept Development Project) في نواة الحلزون ويستمر (تحديث تزايدات متعددة على مسار الحلزون الذي يحيط بالمنطقة المظللة المركزية) حتى يكتمل تطوير المفهوم ، تتقدم عملية البرمجة عبر المكعب التالي (نقطة دخول المشروع تطوير منتج جديد) إذا كان المطلوب تطوير المفهوم ليصبح منتجا حقيقيا وتوضع في بداية لتطوير مشروع جديد، يتطور المنتج الجديد في عدد من المتزايدات حول الحلزون متبعا المسار الذي يحيط بالمنطقة التي لها تظليل اخف من تظليل النواة، ويحدث تدفق مشابه لعملية برمجة أنواع أخرى من المشاريع في الحقيقة، يبقى الحلزون عند تعريفه بهذه الطريقة يعمل حتى نهاية عمر البرمجية (وضعه خارج الخدمة)، هناك أوقات تكون عملية البرمجة فيها نائمة، ولكن حالما يحدث تغيير ما، تبدأ عملية البرمجة عند نقطة بداية مناسبة (مثل تحسين المنتج).

إن النموذج الحلزوني طريقة واقعية لتطوير أنظمة وبرمجيات واسعة النطاق، ولان البرمجيات تتطور مع تقدم عملية البرمجة، فانه من الأفضل للمطور والزبون فهم المخاطر والقيام بالإجراء المناسب لها في كل مستوى من مستويات التطور.

يستخدم النموذج الحلزوني النمذجة الأولية كآلية لتقليل المخاطر، ولكن الأهم من ذلك هو انه يمكن المطور من تطبيق نموذج النمذجة الأولية في أي مرحلة من مراحل تطوير المنتج فهو يحافظ على الطريقة التدريجية النظامية التي تقترحها دورة الحياة التقليدية، ولكن يطبقها بإطار تكراري يعكس واقع العالم الحقيقي. ويتطلب النموذج الحلزوني اعتباراً مباشراً للمخاطر التقنية في جميع مراحل المشروع، وإذا طبق بالوجه المناسب، وجب أن يقلل المخاطر قبل أن تصبح مشكلة يصعب حلها.

مميزات هذا النموذج

- يركز على تقليل المخاطر اثناء عملية التطوير
- يستخدم في المشاريع التي تكون نسبة مخاطرها عالية

عيوب هذا النموذج

- عملية التحكم والضبط صعبة خاصة تحديد حجم وزمن كل دورة
- تقييم المخاطر فيه صارما.

2-5- النماذج الحديثة Modern Models

2-5-1- نموذج المعالجة الموحدة Unified Process

طور قبل حلول عام 1990م ، عندما انتشرت لغات البرمجة الكائنية والتي استخدمت العديد من الكائنات الموجهة في طرق التحليل والتصميم ، حيث تم اقتراح Grady Booch, Ivar Jacobson, and James Rumbaugh والذين هم معا صمموا لغة النماذج الموحدة لتطوير انظمة كائنية عام 1997م Booch, Jacobson, and Rumbaugh later developed قاموا بإضافة المعالجة للغة UML لتصبح ارضية العمل لكينونة هندسة البرمجيات باستخدام لغة UML ولذي استخدمت في تصميم نماذج تصف معمارية البرمجيات. يعتبر نموذج المعالجة الموحدة من احدث النماذج المستخدمة والتي اتصفت بثلاث مناظر رئيسية هي:

- **المنظور المتحرك : تشاهد من خلاله المراحل متحركة مع مرور الزمن**
- **المنظور الساكن : تشاهد من خلاله مراحل المعالجة الساكنة**
- **المنظور التفاعلي: الذي يقترح من خلاله الممارسات الجيدة.**

ويتبع هذا النموذج طريقة انتقال المعالجة التكرارية والتزايدية وبالتالي يكون الشعور تطويري هو الحاصل في المعالجة ، ويقترح هذا النموذج خمسة مراحل لمعالجة البرمجية وهي:

- **1- البدء (Inception,)**
- **2- التعاون elaboration**
- **3- البناء construction**
- **4- الانتقال transition**
- **5- الانتاج and production**

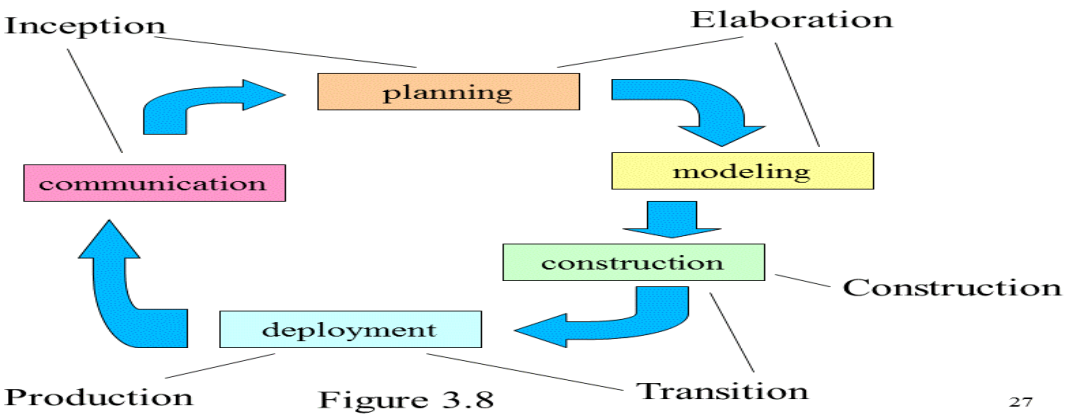
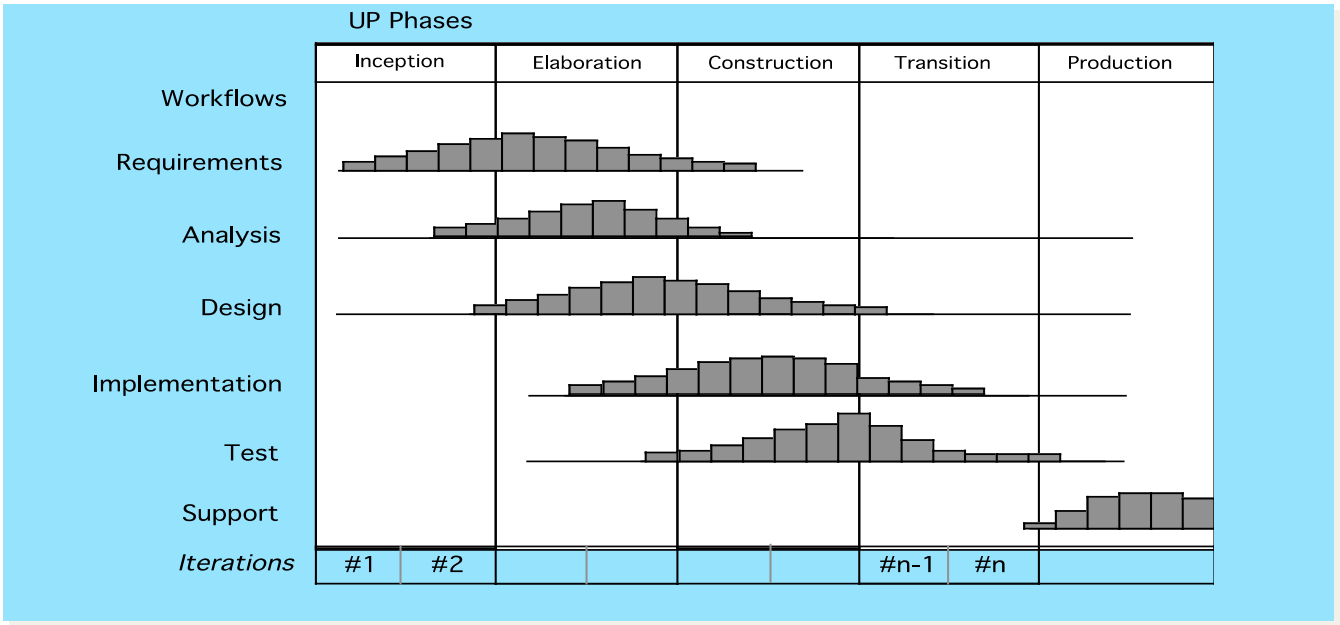


Figure 3.8



2-5-1-1- مراحل نموذج المعالجة الموحدة Phases of the Unified Process Model

1- مرحلة البدء Inception Phase

تشتمل على نشاطات التواصل والتخطيط

2-مرحلة التوضيح Elaboration Phase

تشتمل على نشاطات التخطيط والنمذجة، وتتوسع في تمثيل المعمارية لتشمل خمسة مناظر هي:

- نموذج حالة الاستخدام Use-case model
- نموذج التحليل Analysis model
- نموذج التصميم Design model
- نموذج التطبيق Implementation model
- نموذج الانتشار Deployment model

3-مرحلة البناء Construction Phase

تشتمل على نشاطات البناء

تستخدم نموذج المعمارية السابقة كمدخلات للبناء

يستخدم حالات الاستخدام لاشتقاق اختبارات القبول كمقدمة للمرحلة القادمة

4-مرحلة الانتقال Transition Phase

تشتمل على اخر جزء من نشاط البناء هو او جزء من نشاطات الانتشار

- Encompasses the last part of the construction activity and the first part of the deployment activity of the generic process Software is given to end users for beta(called customer acceptance testing) testing and user feedback reports on defects and necessary changes.
- The software teams create necessary support documentation (user manuals, trouble-shooting guides, installation procedures
- At the conclusion of this phase, the software increment becomes a usable software release

7- مرحلة الانتاجية Production Phase

- Encompasses the last part of the deployment activity of the generic process
- On-going use of the software is monitored
- Support for the operating environment (infrastructure) is provided
- Defect reports and requests for changes are submitted and evaluated

مميزات هذا النموذج

- يعتبر مناسبا لمعظم المشاريع الصغيرة والكبيرة المفهومة والغامضة
- يعتمد على احدث الطرق والتقنيات التي تقوي فاعليته للحلول
- مرونة النظام للتغيرات وسهل الصيانة

عيوب هذا النموذج

- يحتاج الى فريق تقني ذو مهارة عالية لاستخدامه
- يكون مكلفا في بعض الاحيان

نموذج الخفة (الرشاقة) " Agility Modeling "

-النموذج يستجيب للتغييرات بفاعلية (التغيير والتكييف)
-الاتصال الفعال بين كل اطراف النظام(المالكين، المطورين، والمصممين، والمبرمجين، والمستخدمين...)
-تمكين الزبون من مشاركة فريق عمل التطوير
-تنظيم الفريق حتى يتحكم بالعمل المنجز من التطوير ويتحكم بنتائجه.
-الاسراع في التسليم التزايدي للبرمجيات

المعالجة الخفيفة الحركة ،والسرعة An Agile Process

-يتم اشتقاقها من وصف متطلبات الزبون (السيناريوهات)
-تعرف بان الخطط تكون قصيرة الاجل
-تطوير البرمجيات الروتيني (التكراري) مع التاكيد على كل نشاطات البناء.
- التسليم المتعدد من النسخ التزايدية للبرمجيات
-التكيف مع التغييرات التي تحدث

نموذج البرمجة النهائية

Extreme Programming (XP)

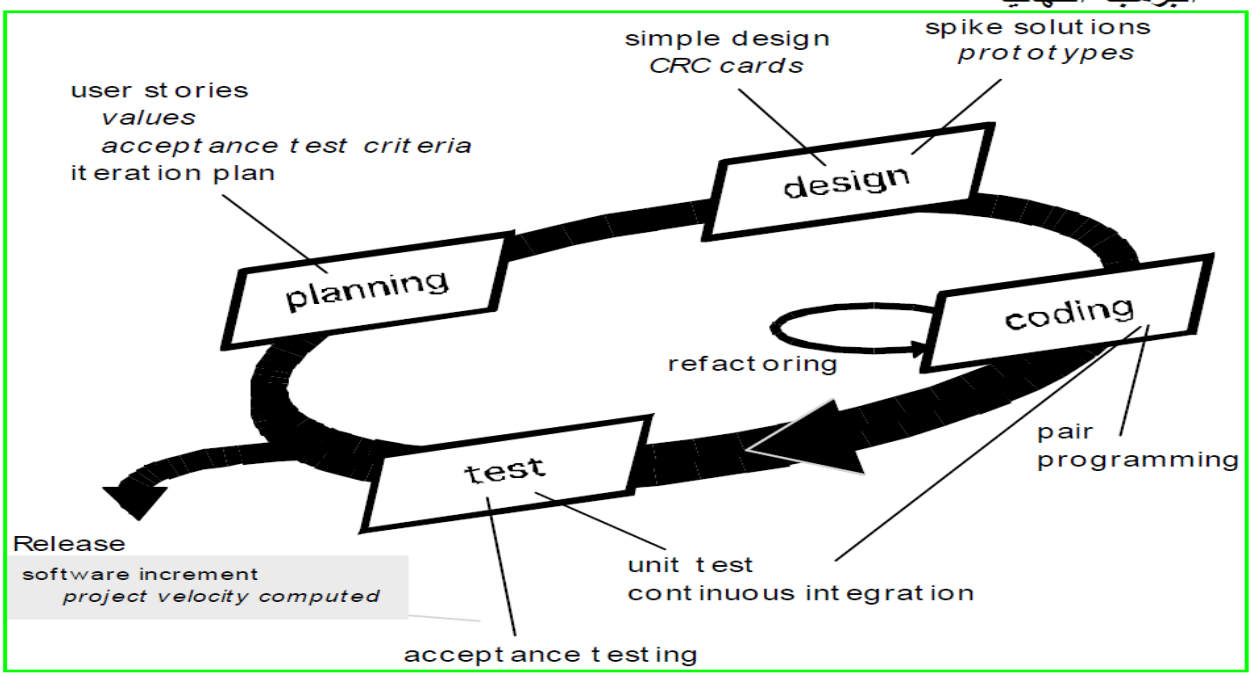
البرمجة النهائية

- The most widely used agile process, originally proposed by Kent Beck
- العملية (المعالجة) السريعة المستخدمة على نحو واسع، اقترحت بالأصل من خلال كنت بك
- XP Planning
- مخطط البرمجة النهائية
  - Begins with the creation of ‘‘user stories’’
  - يبدأ مع أنشاء "قصص المستخدمين"
  - Agile team assesses each story and assigns a cost
  - الفريق السريع يقيم كل قصة ويخصص لها التكلفة
  - Stories are grouped to for a deliverable increment
  - القصص يتم جمعها للتسليم القابلة للزيادة
  - A commitment is made on delivery date
  - التعهد عند تاريخ التسليم
  - After the first increment ‘‘project velocity’’ is used to help define subsequent delivery dates for other increments
  - بعد الزيادة الأولى "سرعة المشروع" تستخدم للمساعدة على تعريف تاريخ التسليم اللاحق للزيادات الأخرى
- XP Design
- تصميم البرمجة النهائية
  - Follows the KIS principle
  - اتبع مبادئ KIS
  - Encourage the use of CRC cards.
  - يساعد (يشجع) على استخدام بطاقات CRC
  - For difficult design problems, suggests the creation of ‘‘spike solutions’’—a design prototype
  - لمشاكل التصميم الصعبة، يقترح إنشاء "حلول المسمار" – تصميم النموذج الأولي.
  - Encourages ‘‘refactoring’’—an iterative refinement of the internal program design
  - يساعد (يشجع) "إعادة التحليل العوامل" – للنقاء التكراري للتصميم الداخلي للبرنامج
- XP Coding
- شيفرة البرمجة النهائية
  - Recommends the construction of a unit test for a store *before* coding commences (Test driven development).
  - يوصي على بناء وحدة اختبار للتخزين *قبل* البدء في التشفير (تطوير قيادة الاختبار).
  - Encourages ‘‘pair programming’’
  - يساعد (يشجع) " البرمجة الزوجية"

- XP Testing
  - All unit tests are executed daily
  - "Acceptance tests" are defined by the customer and executed to assess customer visible functionality
- فحص البرمجة النهائية
  - كل وحدات الفحص تنفذ يوميا
  - "اختبارات القبول" معرّفة من قبل الزبون ونفذت لتقييم وظيفة الزبون المرئية

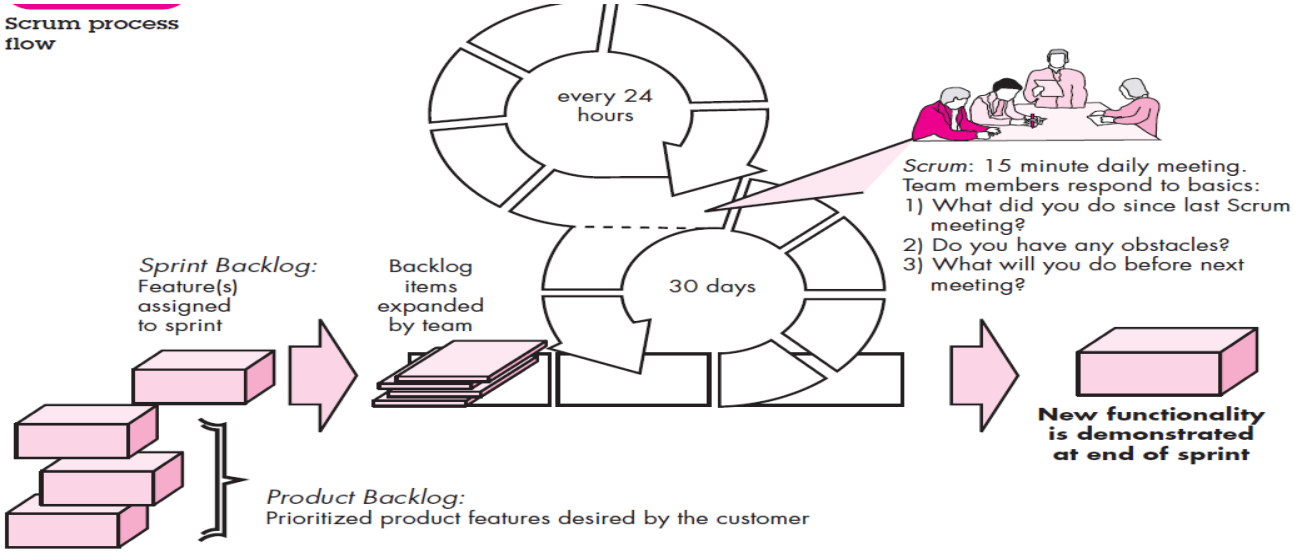
Extreme Programming (XP)

البرمجة النهائية



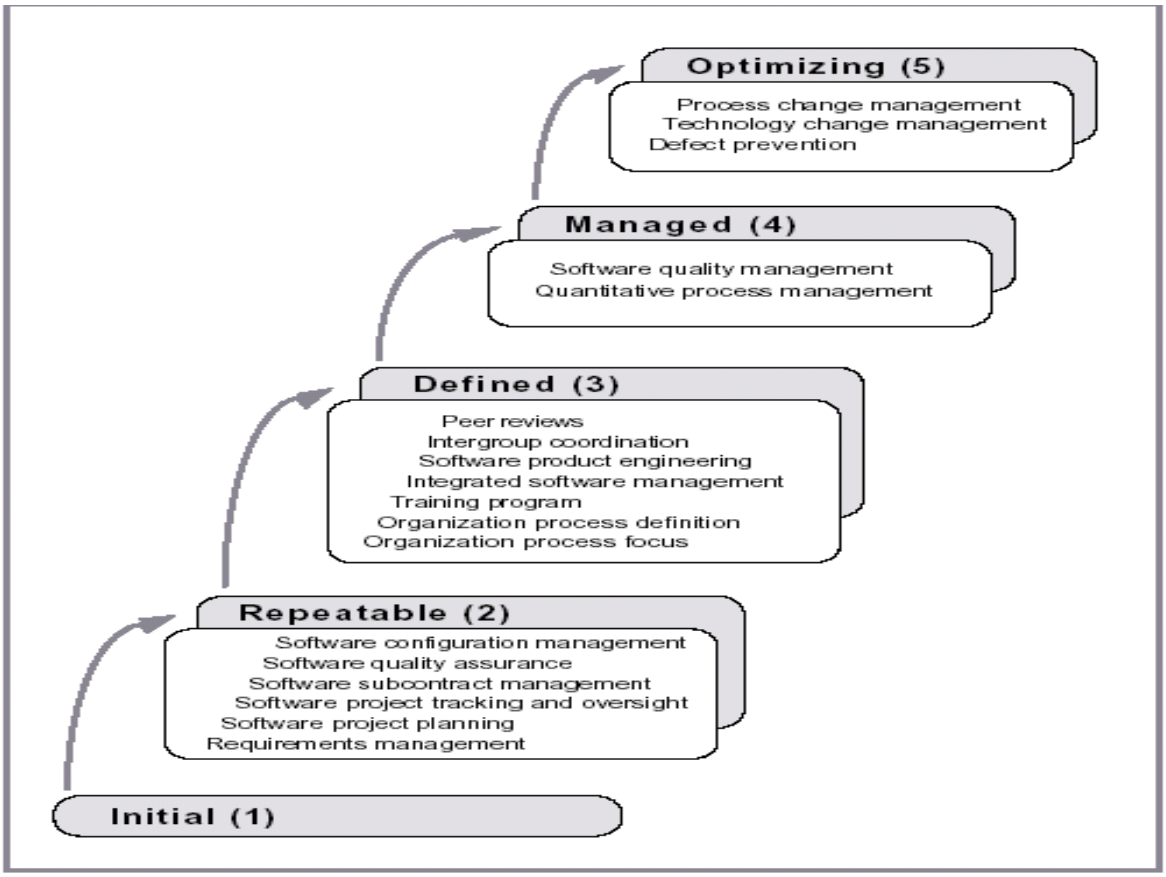
نموذج المزاحمة (التزاحم) Scrum

- Originally proposed by Schwaber and Beedle
- Scrum—distinguishing features
  - Development work is partitioned into "packets"
  - Testing and documentation are on-going as the product is constructed
  - Work occurs in "sprints" and is derived from a "backlog" of existing requirements
  - Meetings are very short and sometimes conducted without chairs
  - "demos" are delivered to the customer with the time-box allocated
- اقترحت بالأصل من خلال سكوبير وبيدل
- الازدحام – الميزات التي تميزه
  - عمل التطوير مقسم إلى "رزم"
  - الاختبار والتوثيق مستمر كبناء المنتج
  - أحداث العمل في "الإسراع" ومشتق من "تراكم" المتطلبات الموجودة
  - الاجتماعات قصيرة جداً أحياناً تجري بدون وجود كراسي
  - "العينات" تسلم إلى الزبون بالصندوق المخصص لها



6-2- نموذج نضج القدرات Capability Maturity Model (CMM)

هنالك إلحاح شديد في السنوات الأخيرة على نضج العملية البرمجية وقد طور معهد هندسة البرمجيات SEI نموذجا شاملا يستند إلى مجموعة من مقدرات هندسة البرمجيات التي يجب أن تكون مكتسبة مع وصول المؤسسات إلى مستويات مختلفة من نضج عملية البرمجة. ولتحديد حالة المؤسسة الراهنة لنضج عملية البرمجة يستعمل معهد الـ SEI استمارة تقييم وسلم تصحيح ذي خمسة مستويات. يحدد سلم التصحيح هذا مدى التوافق مع نموذج نضج القدرات Capability Maturity Model(CMM) ، الذي يعرف نشاطات أساسية مطلوبة على



مستويات مختلفة من نضج عملية البرمجة. يرسم منهج معهد الـ SEI خمسة مستويات لنضج عملية البرمجة تعرّف على النحو التالي:

المستوى الاول - بدائي: توصف عملية البرمجة بأنها منسابة ad hoc وأحيانا فوضوية chaotic .

عمليات برمجة قليلة فقط هي المحدد جيداً، ويعتمد نجاحها على المجهود الفردي.

المستوى الثاني - مكرر: توضع عمليات برمجة أساسية لإدارة المشروع وذلك لمتابعة الكلفة والجدول الزمني والوظائفية، ويكون نظام عملية البرمجة الضروري جاهزا لتكرار النجاحات السابقة التي حصلت في المشاريع ذات تطبيقات مشابهة.

المستوى الثالث- معرّف: تكون عملية البرمجة للنشاطات الإدارية والهندسية موثقة وقياسية ومتكاملة مع عملية البرمجة لكل المؤسسة. تستخدم جميع المشاريع نسخة مصدقة وموثقة من عملية المؤسسة لتطوير وصيانة البرمجيات. يتضمن هذا المستوى جميع الخصائص المعّرفة للمستوى 2.

المستوى الرابع- مُدار: يجمع قياسات تفصيلية لعملية البرمجة وجودة المنتج. وتفهم كل من عملية البرمجة والمنتجات كمّياً، ويتحكم بهما باستخدام قياسات تفصيلية. يتضمن هذا المستوى جميع الخصائص المعّرفة للمستوى الثالث.

المستوى الخامس- مثالي: تتحقق تحسينات مستمرة على عملية البرمجة بتكرار كمية منها، ومن اختبار أفكار وتكنولوجيات مبتكرة. يتضمن هذا المستوى جميع الخصائص المعّرفة للمستوى 4.

ثانيا: المبادئ التي تقود التطبيق (حلول المشاكل)Principles that Guide Practice

2-2-2-1-مبادئ التواصل Communication Principles

المبدئ (1): انصت للمتحدث. Listen.

حاول التركيز الى كلام المتحدث بعيدا عن صياغة استجابتك لما يقوله.

المبدئ (2): عد نفسك قبل التواصل Prepare before you communicate.

ابذل قليل من الجد لكي تفهم المشاكل قبل ان تقابل الاخرين (اصحاب المشروع البرمجي)

المبدئ (3): احد الاشخاص يدير النشاط Someone should facilitate the activity

في عملية المقابلة يجب ان يكون هناك قائد (مسهل الاجراءات) لكي:

(1) يحافظ على توجيه النقاش نحو انجاز المهام

(2) يفض النزاعات التي قد تحدث

(3) ويتأكد من اتباع المباديء العامة في الاجتماع.

المبدئ (4): التواصل وجها لوجه يكون مفضلا Face-to-face communication is best.

المبدئ (5): اكتب الملاحظات ودون القرارات التي تتخذ بين الاطراف Take notes and document decisions

.

المبدئ (6): جاهد من اجل تعاون الجميع Strive for collaboration

المبدئ (7): ركز على مناقشة المواضيع دون تشتت او تنقل Stay focused, modularize your discussion

المبدئ (8): اذ هناك شيء غامض ارسم صورة فكرة اولية فقد يتوضح لاحقا. If something is unclear, draw a picture

المبدئ (9):

ا-بعد الاتفاق على اي شيء تحرك للأمام (a) Once you agree to something, move on;

ب- اذا لم تتفقوا على شيء تحرك الى الامام (b) If you can't agree to something, move on;

جـ- اذا كانت الخصائص او الوظائف غير واضحة ولم تستطيع توضيحها حاليا ايضا تحرك الى الامام.

. (c) If a feature or function is unclear and cannot be clarified at the moment, move on

المبدئ (10): التفاوض لايكون من اجل التنافس او للفوز بلعبة بل يكون التفاوض فاعلا عندما يفوز الطرفان.

Negotiation is not a contest or a game.

2-2-2-2- مبادئ التخطيط Planning Principles

المبدئ (1): Understand the scope of the project

الفهم العميق للمشروع من كل الجوانب قد يرسم خارطة طريق تجنبك من التيهان عند التشعبات الكثيرة ، ويقود فريق البرمجية الى النهاية المحمودة.

المبدئ (2): شارك الزبائن بنشاطات التخطيط للمشروع Involve the customer in the planning activity
لان الزبائن يعرفوا الاسبقية لمطالبهم الملحة ويحددون انواع القيود للمشروع البرمجي

المبدئ (3): اعرف ان الخطة تكون متكررة(متقلبة) Recognize that planning is iterative.
ان خطط المشروع لا تكون كالنحت على الحجارة...!
فعندما يبدا المشروع هناك الكثير من التعديلات التي تجرى عليه وعلى خططه.

المبدئ (4): التقدير يعتمد على ماذا عرفت Estimate based on what you know.
المقصود من التقدير لكي تعطي مؤشر للجهود، والتكلفة ، والزمن للازم للمهام وكلها تعتمد على فهم الفريق الحالي للعمل الذي سيقومون به.

المبدئ (5): خذ بالاعتبار المخاطر عندما تعرف الخطة Consider risk as you define the plan.
لو تعرفت على المخاطر الذي تكون احتمالية قيمته عالية يكون تضمينه في الخطة مهما.

المبدئ (6): خذ راحتك Be realistic.

People don't work 100 percent of every day .

المبدئ (7): اضبط التفاصيل Adjust granularity

المبدئ (8)عرف كيفية ضمان الجودة Define how you intend to ensure quality.

Describe how you intend to accommodate change

المبدئ (9): وضح كيف انت عازم على التأقلم مع التغييرات

Track the plan frequently and make adjustments as required

المبدئ (10): تتبع الخطة بوضوح تام وقوم بالضوابط المطلوبة
المشروع البرمجي يكون متماشي (متوافق) مع الجدل الزمني في كل يوم من ايام تنفيذه .

2-2-2-3- مبادئ النمذجة Modeling Principles

في عمل هندسة البرمجيات ، ممكن الاعتماد على صنفين من النماذج :

1-نماذج المتطلبات(تسمى ايضا نماذج التحليل) *Requirements models (also called analysis models)*
هي تمثل متطلبات الزبون بواسطة إظهار النظام بثلاثة مجالات مختلفة
1- مجال المعلومات 2- مجال الوظائف 2- مجال السلوك.

2-نماذج التصميم *(Design models)*: يمثل خصائص النظام الذي يساعد المطبقون لبنائه بفاعلية : المعمارية ، واجهة المستخدم ، ومستوى الاجزاء والتفاصيل.

2-2-2-3-1- مبادئ نمذجة المتطلبات Requirements Modeling Principles

Principle #1. **The information domain of a problem must be represented and understood.**
كل المعلومات المجمعة عن المشاكل يجب ان تمثل كما تفهم

Principle #2. **The functions that the software performs must be defined .**
الوظائف التي يؤديها النظام يجب ان تكون معرفة

Principle#3.The behavior of the software (as a consequence of external events) must be represented. السلوك للنظام (كالتي تنتج عن احداث خارجية) يجب ان تمثل

Principle #4. The models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.
النموذج الموضح لمعلومات او وظيفة او سلوك يجب ان يقسم بطريقة ليشمل التفاصيل الطبقي(الهيكلي) وفروعه

Principle #5. The analysis task should move from essential information toward implementation detail.

عملية التحليل يجب ان تبدا من المعلومات الاساسية ثم تتوجه نحو تركيب التفاصيل

2-2-2-3- مبادئ نمذجة التصميم Design Modeling Principles

Principle #1. Design should be traceable to the requirements model.

التصميم يجب ان يكون مطابقا لنموذج المتطلبات

Principle #2. Always consider the architecture of the system to be built.

دائما خذ بالاعتبار معمارية النظام الذي تقوم بتصميمه.

Principle #3. Design of data is as important as design of processing functions .

تصميم البيانات يكون مهما كونه يعتبر تصميم الاجراءات الوظيفية للنظام.

Principle #4. User interface design should be tuned to the needs of the end-user. However, in every case, it should stress ease of use.

تصميم واجة المستخدم يجب ان تكون كما يحبذها المستخدم النهائي ولهذا السبب في كل حالة يجب ان تعزز بسهولة الاستخدام.

Principle #5. Component-level design should be functionally independent .

تصميم الاجزاء (المكونات) يجب ان تصمم للأجزاء المستقلة وظيفيها

Principle #6. Components should be loosely coupled to one another and to the external environment.

تصميم الاجزاء يجب ان تفقد الارتباط(الازدواج) بينها البين وبينها وبين المحيط الخارجي.

Principle #7. Design representations (models) should be easily understandable .

يكون تمثيل التصميم للنماذج سهلا جدا ومفهوما

Principle #8. The design should be developed iteratively. With each iteration, the designer should strive for greater simplicity.

التصميم يجب ان يكون تطيره تكراري ، ومع كل تكرار يجب ان يعزز البساطة اكثر.

2-2-2-4- مبادئ البناء Construction Principles

تتضمن نشاطات البناء مجموعة من المهام لكتابة الكود البرمجي والفحص ، والتي تؤدي الى تشغيل البرمجيات وتجهيزها لتسليمها الى الزبون والمستخدم النهائي. مفاهيم ومبادئ كتابة الكود البرمجي تكون قريبة ومتوازية من نمط البرمجة و لغات البرمجة ومن طرق البرمجة

مفاهيم ومبادئ الاختبار تؤدي الى تصميم عمل اختبارات منتظمة لتكتشف مختلف اصناف الاخطاء وباقل التكاليف والجهود.

2-2-2-4-1 مبادئ الاعداد Preparation Principles

Before you write one line of code, be sure you: قبل ان تكتب سطر برمجي واحد كن متأكد انك:

Principle #1: Understand of the problem you're trying to solve. افهم المشكلة التي تحاول ان تحلها

Principle #2: Understand basic design principles and concepts. افهم اساسيات مبادئ التصميم ومفاهيمه

Principle #3: Pick a programming language that meets the needs of the software to be built and the environment in which it will operate.

اقفش لغة البرمجة التي تلبي احتياجات البرمجية التي تبرمجها والبيئة التي سوف تشتغل فيها البرمجية

Principle #4: Select a programming environment that provides tools that will make your work easier. اختار بيئة البرمجة المزودة بالأدوات التي تجعل عملك البرمجي اسهل

Principle #5: Create a set of unit tests that will be applied once the component you code is completed.

قم بتنفيذ مجموعة اختبارات للوحدات التي سوف تستخدم اجزائها مرتا ثنية فور انتهاء البرمجة.

2-2-4-2-2- مبادي كتابة الاوامر البرمجية Coding Principles

عند بداية كتابة الكود البرمجي كون متأكدا انك **As you begin writing code, be sure you:**

Principle #1:Constrain your algorithms by following structured programming practice.

حدد خوارزمياتك باتباع تطبيق البرمجة الهيكلية

Principle #2: Consider the use of pair programming

خذ بالاعتبار استخدام برامج مزدوجة تتبادل المتغيرات

Principle #3: Select data structures that will meet the needs of the design.

اختار هيكلة البيانات التي سوف تواجه (تلبي) كل احتياجات التصميم

Principle #4: Understand the software architecture and create interfaces that are consistent with it.

افهم معمارية النظام وأنشئ واجهات متوافقة معها

Principle #5: Keep conditional logic as simple as possible.

اجعل الشرط المنطقي سهلا قدر الامكان

Principle #6: Create nested loops in a way that makes them easily testable.

أنشئ الدارات (الحلقات) المتقاطعة بطريقة تجعلهم سهل الاختبار

Principle #7: Select meaningful variable names and follow other local coding standards.

اختار معاني ذات دلالة للأسماء والمتغيرات متبعا طرق الكتابة القياسية.

Principle #8: Write code that is self-documenting. اكتب الكود لذي يكون موثقا بنفسه

Principle #9: Create a visual layout (e.g., indentation and blank lines) that aids understanding.

أنشئ ترتيبات مرئية (مسافات بادئة اسطر فارغة) التي تسهل فهم تسلسل البرنامج

2-2-4-3-2- مبادئ التصحيح Validation Principles

Principle #1: After you've completed your first coding pass, be sure you:

- بعد الانتهاء من كتابة الكود كن متأكدا من:

Principle #2: Conduct a code walkthrough when appropriate

-التحكم بكتابة الاوامر البرمجية ومساراتها حتى تكون لائقة بما تقوم به

Principle #3: Perform unit tests and correct errors you've uncovered.

- القيام باختبار كل الوحدات وتصحيح الاخطاء التي لم تكتشفها من قبل الى ان تكون كل الوحدات صحيحة

Principle #4: Refactor the code - اعادة صياغة كتابة الاوامر البرمجية حتى تكون كلها صحيحة

2-2-4-4- مبادئ الاختبار Testing Principles

دافيس يقترح مبادئ الاختبار التالية : **Davis [Dav95] suggests the following**

Principle #1. All tests should be traceable to customer requirements.

كل الاختبارات تكون مطابقة لتحقيق متطلبات الزبون.

Principle #2. Tests should be planned long before testing begins .

الاختبارات تكون مخططة من قبل الشروع بها.

Principle #3. The Pareto principle applies to software testing.

موضة المبادئ تستخدم لاختبار البرمجيات

Principle #4. Testing should begin "in the small" and progress toward testing "in the large". الاختبار يجب ان يبدا بالجزئيات الصغيرة ويتطور الى الكبيرة

Principle #5. Exhaustive testing is not possible. الاختبار العشوائي (الاعمى) ليس ممكنا

2-2-5- مبادئ الانتشار Deployment Principles

Principle #1. Customer expectations for the software must be managed.

1- توقعات الزبون عن النظام يجب ان تتحقق تماما. احيانا كثيرة يتوقع الزبون اكثر مما اعده فريق التطوير للتسليم ، ويحدث سوء تفاهم حالا.

Principle #2. A complete delivery package should be assembled and tested.

2- الحزمة المتكاملة للتسليم يجب ان تجمع وتختبر جيدا

Principle #3. A support regime must be established before the software is delivered. An end-user expects responsiveness and accurate information when a question or problem arises.

3- الدعم الصحي يجب تأسيسه قبل تسليم النظام فالمستخدم النهائي يتوقع استجابات سريعة لتصحيح المعلومات عندما يسال او تتواجد مشاكل.

Principle #4. Appropriate instructional materials must be provided to end-users.

4- كل اوامر التعليمات اللازمة للنظام يجب منحها للمستخدم النهائي

Principle #5. Buggy software should be fixed first, delivered later.

5- النظام المصاب (المعطوب) يجب تصحيحه ومن ثم تسليمه فيما بعد.

رابعا: تقنيات الحاسب تساعد هندسة البرمجيات (Computer-Aided- Software Engineering (CASE Tools

هي عبارة عن مجموعة من البرمجيات المعدة بإحكام لدعم تطوير البرمجيات (معالجة التحليل ، التصميم ، توليد الكود البرمجي ، الفحص والاختبار ، الصيانة) وارتقائها .

مثلا: حزمة البرامج .NET , IDE.

برامج مساعدة لتوليد الكود البرمجي

برامج محررات الرسومات المستخدمة في تطوير البرمجيات تحرير واجهة الاستخدام ، تصميم نماذج التحليل ، نماذج التصميم ، رسم الكائنات الموجهة ، رسم مخططات انسياب البيانات، رسم السيناريو ، والممثل ، وحالات الاستخدام ..، وبرامج البناء لتوليد الكود البرمجي ... مثلا:

1- حزمة البرامج .NET , IDE ، UML, Eclipse, Bean, MS-Viso, Rational Rose

2- ادوات تصمم قاموس البيانات (OO Design Tool)

3- برامج الفحص والاختبار Debugging Software

4- برامج تصميم الويب Web Developer

5-برامج تصميم وتخطيط قواعد البيانات Oracle Developer

6- برامج برمجة وتوزيع وتصميم الشبكات

4- برامج بناء المترجمات Compiler Design

5- برامج الصيانة

6- برامج نمذجة الاعمال والشركات ومعالجة الاجراءات MS- VISO

رغم فعالية هذه الادوات والبرامج والتي سهلت كثير من عمليات هندسة البرمجيات المعقدة ، الا انها مازالت قاصرة امام الكثير من الصعوبات كونها تستخدم فقط للاستفادة من طرقها في الحل او لمعرفة جوانب الغموض في تحليل خوارزمية ما او تصميم اجراءات معقدة ولا يعمل بها الا بعد التحديث والتنقيح .. فهندسة البرمجيات تحتاج الى فكر مبدع وهذا يصعب اتمته رغم كل المحاولات الحديثة.

للمشاريع الكبيرة :هندسة البرمجيات تحتاج الى نشاطات مكثفة من فرق تقنية متخصصة ذات مهارات عالية ، خيال واسع -اخلاص للمهنة -وافكار تمزج وتعاون مشترك- ونيات صادقة ، وجهود واموال تصرف في التفاعلات مع بعضها من اجل النجاح ، وبعض الادوات يمكن ان تساعد في اجرا بعض النشاطات كوسائط للترتيب والتوضيح والتواصل وكوسائل تختصر الجهد والوقت لكنها لا تحل محل فكر العقل البشري وابداعاته.

الفصل الثالث

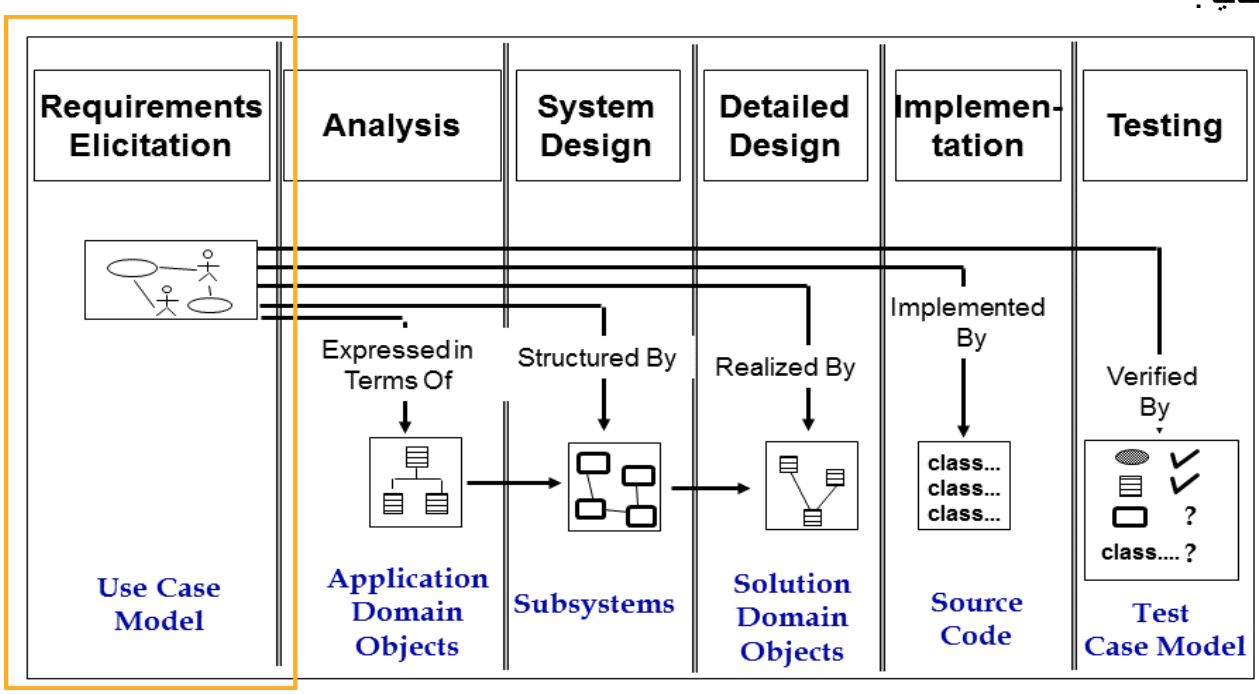
فهم المتطلبات

Understanding Requirements

1- معنى المتطلبات Meaning Requirements

تبين مجموعة المتطلبات ما يجب أن يقوم به النظام وتقوم بتعريف القيود على تشغيل هذا النظام وتنفيذه، وتبين مجموعة المتطلبات الوظيفية الخدمات التي يوفرها النظام، وتبين مجموعة المتطلبات غير الوظيفية القيود التي يتم خلالها تطوير النظام.

هندسة البرمجيات عمل إبداعي يتم خطوة بخطوة بتعاون فريق عمل لكل منهم مهمة محددة وباستخدام طرق معينة لتنظيم المعالجات ، وتمر عملية بناء أي منتج برمجي بعدة مراحل يطلق عليها أسم دورة الحياة وتتضمن الأنشطة التالية:



- تحديد وتعريف المتطلبات.
- تحليل النظام
- تصميم النظام.
- تصميم المحتويات (الاجزاء)
- التنفيذ (كتابة اكواد البرامج).
- اختبار البرنامج واختبار النظام.
- تشغيل وصيانة النظام.

2-3- خطوات تحديد المتطلبات

هناك العديد من الاجراءات والخطوات التي يقوم بها مطورو النظام لتحديد المتطلبات وقد تنقسم طرق التحديد الى طريقتين هما:

1-الطرق البدائية في استنتاج المتطلبات

2-الطرق الحديثة في استنتاج المتطلبات

3-2-1- الاجتماعات مع العميل للتعرف على متطلباته.

3-2-2- تسجيل المتطلبات فى وثائق أو قاعدة بيانات، وعرضها على العميل ليوافق عليها باعتبار أنها ما يطلبه بالفعل.

3-2-3- إعادة تسجيل المتطلبات رياضياً ليتمكن المصمم من تحويلها إلى تصميم جيد للنظام.

3-2-4- استخدام هندسة المتطلبات.

3-2-5- تأسيس ارضية العمل للمشروع Establishing the Groundwork for Project

3-2-6- التثبت والتحقق من المتطلبات.

3-3- أنواع المتطلبات Types of Requirements

3-3-1- متطلبات المستخدم: متطلبات المستخدم عبارة عن جمل بلغة طبيعية مع أشكال توضيحية للخدمات التى يوفرها النظام وقيود التشغيل مكتوبة للمستهلك.

3-3-2- متطلبات النظام: هي وثيقة هيكلية تبين الوصف التفصيلي لخدمات النظام، مكتوبة كعقد بين المقاول والمستهلك.

3-3-3- مواصفات البرمجيات: صفات او وصف تفصيلي للبرمجيات تعمل كأساس لتصميم وتنفيذ، مكتوب لمطوري النظام (المهندسين والمصممين والبنائين والفاحصين الذين يحتاجون معرفة ما سيقوم به النظام بدقة).

3-4- وتقسم المتطلبات من حيث طبيعتها الى المتطلبات الوظيفية وغير الوظيفية Functional and non functional Requirements

3-4-1- المتطلبات الوظيفية: يجب توفير بيان وإفادة وعرض الخدمات التى يوفرها النظام، وكيف يجب للنظام أن يتعامل مع المدخلات المعينة ، وكيفية تصرف النظام فى مواقف معينة.

تختص المتطلبات الوظيفية بالخصائص التالية:

1- تصف وظائف وخدمات النظام.

2- تتعلق بنوعية البرامج والمستخدمين المتوقعين وطبيعة العمل الذي سيُستخدم فيه النظام.

3-4-2- المتطلبات الغير وظيفية:

تعريف خصائص النظام والقيود: مثل سهولة الاستخدام ،الموثوقية، الاعتمادية ، الرصانة ، والامانة ،والاداء ، وزمن الاستجابة ، قابلية القياس، والانتاجية ، المرونة ، قابلة التكيف ، قابلية النقل ، قابلية الصيانة.

التصنيفات الغير وظيفية Non functional Classifications

متطلبات المنتج: وهى متطلبات مواصفات تعرف المنتج المسلم بطريقة معينة مثل سرعة التنفيذ والاعتمادية ومتطلبات قابلية النقل ومتطلبات قابلية الاستخدام ومتطلبات الكفاءة ..وغيرها.

المتطلبات التنظيمية: وهى متطلبات تعبر عن نتائج سياسات المنظمة والإجراءات مثل المعايير المستخدمة في المنظمة ، ومتطلبات التسليم والتنفيذ وغيرها.

المتطلبات الخارجية: وهى متطلبات تنشأ من عوامل خارج النظام وعملية تطويره مثل متطلبات قانونية ومتطلبات السلطة التشريعية وهي (متطلبات قابلية النظام ومتطلبات اخلاقية ..وغيرها).

3-4-2- متطلبات المجال: Domain Requirements

المتطلبات التي تأتى من مجال التطبيق للنظام والخصائص المنعكسة من هذا المجال ، ومستخرج من مجال تطبيق ووصف خصائص النظام والملامح التي تعكس المجال وقد تكون متطلبات وظيفية جديدة أو قيداً على متطلبات موجودة أو تعريف عمليات حسابية معينة.

مشاكل متطلبات المجال:

قدرة الفهم: فالمتطلبات يعبر عنها بلغة مجال التطبيق وهذا غالباً ما يكون غير مفهوم لمهندس البرمجيات المطور للنظام.

الوضوح: متخصص المجال يفهم المنطقة بدرجة جيدة للدرجة التي لا يقوم فيها بالتفكير فى جعل متطلبات المجال واضحة بينة.

3-4-3 الأهداف والمتطلبات Goals and Requirements

لا يمكن تحديد المتطلبات الغير وظيفية بدقة كما أن المتطلبات الغامضة يصعب تحقيقها.

الهدف: مقصد عام يتحقق للمستخدم مثل سهولة الاستخدام.

تساعد الأهداف المطور على بلوغ وتحقيق رغبات ومقاصد مستخدمي النظام.

تفاعل المتطلبات:

التنازع بين المتطلبات الغير وظيفية المختلفة أمر شائع فى النظم المعقدة.

3-5-الخطوط الرئيسية لكتابة المتطلبات

1- اعتمد صيغة معيارية لجميع المتطلبات.
2- استخدم اللغة الطبيعية بطريقة متجانسة.
3- استخدم وسائل إظهار النص كالكتابة بلون غامق لتحديد الأجزاء الهامة من المتطلبات.
4- تجنب استخدام الاختصارات في كتابة المتطلبات.

عند كتابة المتطلبات يجب ان تخصص المتطلبات حسب مهامها الى مايلي:

3-5-1-متطلبات النظام System Requirements

مواصفات أكثر تفصيلاً من متطلبات المستخدم، تخدم أسس تصميم النظام، وقد تستخدم كجزء من العقد المكتوب المحرر بين المطور والمستخدم أو مستهلك النظام، وقد يعبر عن متطلبات النظام باستخدام نماذج النظام.

3-5-2- اللغة المهيكلة للمواصفات Structured Language Specifications

استخدام مواصفات مبينة على شكل ثابت مثل اللغات المهيكلة الغير قابلة للتفسير والتأويل.

تعريف الوظيفة أو المكون، وصف المدخلات ومن أين تأتى، وصف المخرجات وإلى أين تذهب، الإشارة إلى أي مكونات أخرى مطلوبة، الشروط، والتأثيرات الجانبية إذا كانت موجودة.

3-5-3- مواصفات واجهة المستخدم Interface Specification

أنواعها:

- الواجهات الإجرائية
- هيكل (تركيب) البيانات التى سيتم تبادلها
- تمثيل البيانات
- التدوين الشكلى طريقة تقنية مؤثرة لمواصفات واجهة المستخدم.

3-6- عمليات هندسة المتطلبات Requirements Engineering Processes

هندسة المتطلبات: هى عملية إنجاز الخدمات التي يطلبها الزبون من النظام والقيود التي يعمل ويطور فيها النظام، المتطلبات نفسها هى وصف لخدمات النظام والقيود التي تتولد خلال عملية هندسة المتطلبات.

إن الهدف النهائي من هندسة المتطلبات هو توصيف متطلبات البرمجية (SRS) Software Requirements Specification من خلال كتابة وثيقة المتطلبات.

توصيف البرمجيات هي الإجرائية التي تسمح بتحديد الخدمات المطلوبة من النظام، والقيود التي تقيد تطويره وتشغيله. إن هذا النشاط يسمى هندسة المتطلبات .

تتألف إجرائية هندسة المتطلبات من الخطوات التالية:

- ❖ تجميع المتطلبات وتحليلها.
- ❖ التحقق من صلاحية المتطلبات.
- ❖ توصيف المتطلبات.

توصيف المتطلبات يتم بعد سلسة من التحقيقات والاستشارات والتنقيحات والاضافة والحذف لمسودة المتطلبات ،وتجرى المداولات بين المطورين والزبون للاتفاق على نسخة نهائية تسمى "توصيف المتطلبات" وهذه منها تصاغ بنود عمل عقد التطوير للمشروع البرمجي ، واذا كانت التوصيفات كبيرة فيجب تعريف الاحتياجات بملخص مجرد بطريقة لا تحتاج إعادة تعريف او تأويل اخر، ويجب أن تكتب المتطلبات حتى يتمكن المطورون من استبيان العقد والعروض وعرض احتياجات الزبون (العميل) بطرق مختلفة، ويعد ان يتم كتابة العقد حتى يجب على مهندس البرمجية أن يقوم بكتابة تعريف النظام للمستهلك بتفاصيل أكثر حتى يتمكن من الفهم ويتحقق مما تفعله البرمجية عند التسليم ، وكلا من هذين المستندين هما مستندات متطلبات النظام.

- **تتضمن هندسة المتطلبات:** دراسة الجدوى واستنباط وتحليل المتطلبات ومواصفات وإدارة هذه المتطلبات.
- تختلف مناهج تقسيم مهام هندسة البرمجيات لكنها تتفق فى النهاية على مجمل العمليات التى تتم فيها وهي كالاتي:

1- الانطلاق او البدء Inception

-الانطلاقة تبدأ بجمع المعلومات حول مشكلة المؤسسة المحتاجة لتطوير نظام الي.

-قوم بتعريف كل الاطراف المستفيدة من النظام ، وتحديد الاشخاص الذين يريدون حل المشكلة ، اجمع المعلومات حول المشكلة.

- يتم توجيه مجموعة اسئلة للزبون وشركائه وموظفيه لتكون اساسا لفهم المشكلة التي نحن بصدد تحليلها.

-التعرف على مختلف وجهات النظر المتعددة لشركاء العمل في مؤسسة الزبون.

-تعزيز العمل التعاوني بين الشركاء في العمل ، وتركيز الأسئلة على الزبون وشركائه وموظفيه ومناقشة الهدف الكلى من النظام والمنافع المتوقعة لكل الاطراف.

-تحديد اللاعبون الرئيسيون للمشروع : Identify stakeholders

- من هو باعتقادك ايضا مهم يجب علينا التحدث معه.

-التعرف عل كل الاطراف والجوانب المحيطة بالمشروع

- دفع العمل التعاوني الى الامام(حث فريق المشروع تجاوز الصعاب والتقدم الى الامام)

- اسأل الاسئلة الاولى؟ The first questions?

- من هو الشخص المطالب لهذا العمل؟
- من الشخص الذي سوف يستخدم هذا الحل؟
- ماهي الفائدة الاقتصادية من وراء نجاح هذا الحل؟
- هل هناك مصادر للحل ونحن نحتاجها؟
  - ثم ناقش طبيعة الحلول المقترحة معهم لهذه المشكلة.
  - صيغ المصطلحات الاولية عن خصائص المشكلة.
  - استخدم نماذج حالة الاستخدام وانسياب الاحداث بينها لتحديد المتطلبات.
  - ارسم واجهات استخدام ولو يدوية (Sketch).

- استخدم نموذج اولي (اختياري) للتوضيح

2- استخلاص المتطلبات Eliciting Requirements

– يتم استخلاص المتطلبات من كل الاطراف المستفيدة من النظام والتي هي صاحبة القرار ومؤثرة في مكوناته ومواصفاته ووظائفه.

-تحديد المقابلات بحيث تكون بين الطرفين مهندسي البرمجيات(المطورين للبرمجيات) و(الزبائن والمستخدمين).

- قوانين الإعداد للمقابلة والمشاركة تكون مستخدمة ومتبعة.

- اقتراح جدول الاعمال للمقابلة قبل انعقادها.

-المنظم للمقابلة ممكن ان يكون المطور للنظام او الزبون او اي شخص مناسب من خارج المؤسستين يقوم بإدارة المقابلة.

-الية التعريف ممكن تكون ورقة عمل او لوحة واقفة على قوائم او لوحة لاصقة على الحائط او لوحة الكترونية او غرفة محادثة او اي وسائط عرض مثل عارض البيانات.

- الهدف من المقابلة يكون: The goal of meeting is:

- لمعرفة المشاكل والاحتياجات To identify the problems
- تقديم العناصر المقترحة للحلول Propose elements of the solution
- اجراء المفاوضات بين كل الاطراف ومختلف الآراء Negotiate different approaches, and
- تحديد مجموعة الحلول للمتطلبات Specify a preliminary set of solution requirements

3- توسع المتطلبات: Elaboration Requirements

– انشاء نماذج التحليل التي تعرف كامل البيانات والوظائف وسلوك المتطلبات وخصائصها.

– تنقيح المتطلبات المجمعة وتصويبها لبناء وتحقيق معمارية النظام.

– تحديث صيغ المصطلحات لتشمل معظم متطلبات البرمجية

– اجراء توصيف حالة الاستخدام المضافة.

– تنقيح حالة الاستخدام المطورة في التكرار السابق.

– قرر حتى تكون حالة الاستخدام منظورة من المعمارية للنظام

4- التفاوض: Negotiation

يكون التفاوض من اجل نجاح النظام وتسليمه وكتابة العقود والذي يجب ان يكون مرن ومراعي لمصلحة المطورين والزبائن ولا يكون التفاوض من اجل فوز طرف على الاخر.

5- توصيف المتطلبات: Specification Requirements

يكون توصيف المتطلبات واحدة او اكثر من الخطوات التالية:

- يكون التوصيف كتابة الوثيقة للنظام والمشتملة عن كل مواصفاته.
- يكون التوصيف مجاميع من احداث المستخدمين وادوارهم وحالات الاستخدام للنظام
- يكون التوصيف مجموعه من النماذج التي توضح وتحدد عمل النظام.
- يكون التوصيف شاملا الصيغ الرياضية والحسابية المستخدمة بالنظام.
- يكون التوصيف عبارة عن نموذج اولي يوضح كيف سيكون النظام بعد تطويره.

6- التحقق من المتطلبات: Validation Requirements

هي عبارة عن الية المراجعة الشاملة التي تركز على:

- الاخطاء بالمحتويات او اخطاء التنفيذ والتفسير

- المناطق التي تحتاج لتوضيح المتطلبات
- وقد تكون المراجعة لمعرفة فقدان بعض المعلومات

مراجعة كامل اجزاء ومكونات النظام لاجتثاث بعض المشاكل التي ترافق هندسة منتجات برمجية كبيرة

- مراجعة الاختلاف او عدم الوثوق (نتيجة عدم تحقيق) المتطلبات

7- ادارة المتطلبات: Requirements management

ادارة المتطلبات هي مجموعة من الأنشطة التي تساعد فريق المشروع على التعريف والتحكم والمتابعة لتنفيذ المتطلبات والتغييرات الناتجة كلما تقدم المشروع، ومعظم هذه الأنشطة تماثل تلك التي يتم عملها في عملية إدارة تكوين البرمجيات. يتم تعريف المتطلبات أولاً وتسمى بالنوع (وظيفياً وبيانات وسلوك وواجهة أو مخرجات) يتم تطوير جدول التعقب مثل الملامح والمصدر والاعتماد والنظم الفرعية من الواجهة الرئيسية، ويتم تحديثها في أي وقت يتم فيه تعديل المتطلبات عليها اثناء تطوير المشروع .

8- تأسيس ارضية عمل لمشروع برمجي Establishing the Groundwork

لكي يتم فهم متطلبات المشروع البرمجي –لكي تجد بداية الطريق التي تمكنك من التقدم نحو الحل الناجح لابد وان يتم تأسيس ارضية عمل لهذا المشروع المزعوم انشاؤه تلتزم بتنفيذ الخطوات التالية:

1- البدء (الانطلاقة) : Inception

تم شرحها في هندسة المتطلبات (هي نفسها)

2- استخلاص المتطلبات: Eliciting Requirements

تم شرحها في هندسة المتطلبات (هي نفسها)

3- عملية نشر الجودة Quality Function Deployment

تجهيز وظيفة الجودة بتعريف ثلاث أنواع من المتطلبات العادية والمتوقعة والموجودة، وفى اجتماعات الزبون(العميل) يستخدم تجهيز الوظيفة لتحديد قيمة كل وظيفة مطلوبة للنظام، ويعرف تجهيز المعلومات كلا من كائنات البيانات والأحداث التي يجب على النظام أن يستهلكها أو ينتجها.

4- استنباط عمل المنتجات Elicitation Work Products

- بيانات الاحتياجات والجدوى.
- بيان مقيد عن مدى النظام أو المنتج.
- قائمة بمجموعة الشركاء المنشغلين باستنباط الاحتياجات.
- وصف بيئة النظام الفنية.
- قائمة من المتطلبات منظمة بواسطة الوظائف وقيود المجال المطبقة.
- نماذج أولية يتم تطويرها للفهم الأفضل للمتطلبات.

9- نماذج تحليل المتطلبات Requirements Analysis Models

تستخدم نماذج تحليل المتطلبات لتوفير توصيفات للمعلومات المطلوبة والوظائف والمجالات السلوكية للنظم المبنية على الحاسوب ، وهناك العديد من النماذج التي تساعد على تحليل المتطلبات واظهار وظائفها وخصائصها حتى يسهل تنقيحها واثباتها ومن عناصر نماذج التحليل مايلي:

- العناصر المبنية على السيناريو: وتصف النظام من وجهة نظر المستخدم.

- العناصر المبنية على الفئة: تبين العلاقات بين الكائنات التى تدار بواسطة الفعل وسماته.
- العناصر السلوكية: تصور النظام وتصرف الفئة كحالات وانتقالات بين الحالات.
- العناصر التدفقية المنحى: تبين كيفية انسياب البيانات من خلال النظام وهل تم تحويلها بواسطة وظائف النظام.

10- توثيق المتطلبات Requirement Documentation

- وثائق المتطلبات هى مسودة رسمية وافادة بما هو مطلوب من مطور النظام، ويجب أن تتضمن كلا من تعريف المتطلبات ومواصفاتها، وهى ليست مستند تصميم، وبقدر الإمكان فإنها عبارة عن مجموعة مما يجب أو يجب أن يفعله النظام بدلا من أن تكون مجموعة من كيفية ما يجب عمله.
- هيكل توثيق المتطلبات Requirements Document Structure
وتتضمن مقدمة، مفردات، تعريف متطلبات المستخدم، بنية النظام، مواصفات متطلبات النظام، نماذج النظام، ارتقاء النظام، ملاحق، وفهرس.

1-10 وظائف وثيقة المتطلبات: Roles of the SRS

- هي اساس التواصل بين كل الاطراف.
- هي المدخلات لفريق التصميم
- هي المدخلات لفريق الاختبار وضمان الجودة.
- المدخلات لتوثيقات المستخدم.
- ميثاق تعاقدي بين الاطراف المطورة والمستخدمة للنظام.
- المرجع لمدير البرمجيات.
- التحكم بتطوير النظام.

2-10 مكونات وثيقة المتطلبات Software Requirements Specification (SRS)

ان توصيف متطلبات البرمجيات (SRS) هي عبارة عن وثيقة تتكون من مواصفات النظام التفصيلية ، ولكل مظاهر المشروع المزعوم تطويره والتي يجب ان تكون موجودة مع كل ملاحظات المشروع البرمجي .

وبالحقيقة لا تكتب مرة واحدة بل انها تنقح الى عدة نسخ تكون اخرها هي المتفق عليها بين كل الاطراف انها وثيقة توصيف متطلبات النظام توزع لكل الأطراف المستفيدة والمشاركة في تطوير المشروع البرمجي.

متطلبات النظام توزع لكل الأطراف المستفيدة والمشاركة في تطوير المشروع البرمجي.

11- قالب وثيقة المتطلبات :

الجزء التمهيدي للمشروع

الجزء التمهيدي للمشروع: يركز الجزء التمهيدي من وثيقة المتطلبات على المدراء وصانعي القرار واللذين لاير غبون في التفاصيل.		
1	هدف المشروع ونطاقه	يتم في هذا الجزء وضع اهداف المشروع وطريقة العمل بتحقيقها. والاشارة الى نطاق المشروع وتعريف حدوده ، وامكانياته وعدم امكانياته.

2	**سياق العمل**	تصف وثيقة المتطلبات حالة عمل النظام وتوضح كيف سيقوم بتحقيق اهداف وغايات المؤسسة المحتاجة له.
3	**الاشخاص المعنيون**	يتم هنا تعريف جميع الاشخاص ذوي الصلة بالمشروع وذكر اسمائهم مباشرتا. وتوضيح ادوارهم ومهامهم.
4	**افكار العمل**	يتم هنا الاشارة الى بعض افكار الحل والاهتمام بالحلول البرمجية الجاهزة فشراء اجزاء او برامج جاهزة والتعديل عليها افضل بكثير من تطوير كامل اجزاءه من نقطة البداية، مع توضيح الاجزاء التي ستستخدم جاهزة ، مع تحري دقة ومدي صلاحيتها وتوافق واجهاتها في الاستخدام.
5	**نظرة عامة للوثيقة**	اخيرا من الجيد ان تذكر بقية اجزاء الوثيقة هنا بالجزء التمهيدي حتى تسهل فهم محتوياتها ، كما يفضل ان توضح النظرة العامة لمراحل التحليل والتصميم التي يتبعها المطورون.
خدمات النظام:		
1	**نطاق النظام**	يتم تخصيص الجزء الاكبر من وثيقة المتطلبات لخدمات النظام وهو الجزء العملي الذي يحتوي على نماذج متطلبات العمل.
2	**متطلبات النظام**	يتم رسم مخطط انسيابي للنظام يبين فيه تدفق البيانات DFD بشكل محدود حتي تتضح نطاق وحدود المشروع وتسهل عملية اتساع او تخفيض هذا النطاق.
3	**متطلبات المعطيات**	ويمكن نمذجة متطلبات الوظائف بمخطط حالات استخدام العمل (Use Cases Diagram). سيتوضح هذا لاحقا ، ويمكن نمذجة متطلبات المعطيات من خلال مخطط كائنات (Classes Diagram) ، سيتوضح هذا لاحقا.
قيود النظام: تعتبر قيود النظام كل الحالات والاجراءات التي تؤدي الى تقييد عمل النظام ومنعه من القيام بخدماته او تطويرها ، وتحدد قيود النظام بما يلي:		
1	**متطلبات الواجهة**	تعرف قيود الواجهة كيف يتخاطب المنتج مع المستخدمين، فنعرف في وثيقة المتطلبات المظهر العام لواجهات الاستخدام البيانية فقط، اما التصميم الاولي لهذه الواجهات فيتم خلال مرحلة توصيف المتطلبات لاحقا اثناء تصميم النظام.
2	**متطلبات الاداء**	تعرف قيود الاداء عن السرعة التي يجب ان ينجز فيها النظام مهامه المختلفة. او بطريقة اخر زمن استجابة النظام. كما يمكن ان تشمل هذه المتطلبات على قيود اخرى تتعلق مثلا بوثوقيه النظام وجاهزيته للعمل الدائم وغيرها.
3	**متطلبات الامن**	تصف قيود الامن والحقوق والصلاحيات الممنوحة للمستخدمين للوصول الى المعلومات تحت سيطرة النظام، فقد يعطي المستخدم امكانية مقيدة للوشول الى المعلومات او لحقوق معينة لتنفيذ عمليات محددة على المعطيات.
4	**متطلبات التشغيل**	تحدد قيود التشغيل للبنية العتادية والبرمجية التي سيعمل ضمنها النظام، وقد يكون لهذه المتطلبات اثر مباشر على مواضيع اخرى ذات صلة بالمشروع. كتدريب المستخدمين او صيانة النظام.
5	**متطلبات سياسية او قانونية**	تعبر القيود السياسية والقانونية عن امكانية او عدم امكانية توزيع المنتج لاسباب سياسية او قانونية وغالبا ما تعتبر معروفة ضمنيا بحيث لا تتم الاشارة اليها صراحة في الوثيقة.
6	**قيود أخرى**	قيود غير محددة قد تظهر من خارج المؤسستين البائعة والمالكة للنظام.
مواد المشروع: يتم تعريف كل ما يمكن ان يؤثر على نجاح المشروع ولم نتطرق لذكره في موضع اخرمن الوثيقة		

1	**مواضيع مفتوحة**	يمكن ان يتضمن ذلك النمو المتوقع لأهمية بعض المتطلبات التي تقع حاليا خارج نطاق المشروع كما يمكن ان يتضمن اي مشاكل محتملة يمكن ان تظهر عند تسليم النظام. يمكن استخدام مخططات Pert Chart، Ghant وغيرها في ادارة المشروع ووضع خطط قياسية.
2	**الجدول الزمني الاولي**	الجدول الزمني الاولي يتم فيه تحديد الفترة الزمنية الاولية التقديرية للمشروع ، بحيث يشمل هذا تحديد الكادر البشري اللازم وتحديد الموارد المطلوبة، وتحديد الفترات اللازمة للتطوير، المواد الاخرى اللازمة للمشروع .
3	**الموازنة الاولية**	في البند «موازنة اولية» يتم وضع التكلفة الاولية للمشروع واحتساب قيمتها اعتمادا على الجدول الزمني المحدد سابقا.
الملحقات: يمكن ان يتضمن هذا الجزء من وثيقة المتطلبات معلومات اخرى تكون مهمة ومفيدة والتي يتم جمعها وتحديديها لفهم المتطلبات ويمكن ان تكون الملحقات كما يلي:		
1	**معجم المصطلحات**	يتم فيه تعريف المفردات والاختصارات المستخدمة في وثيقة المتطلبات.
2	**استمارات ووثائق العمل**	وفيه يتم جمع وادراج كافة وثائق واستمارات العمل المستخدمة او المملوءة بالمعلومات. وارفاق النماذج المختلفة للإجراءات المختلفة نماذج (المخرجات ، المعالجة ، المدخلات)، نماذج تعليمات استخدام النظام ، نماذج خطوات الدراسة والتقارير والمحاضر الهامة.
3	**المراجع:**	وفيها يتم تحديد الوثائق والمصادر التي استخدمت في جمع المتطلبات مثل مواقع الانترنت او الكتب او المجلات وغيرها.

12 التفاوض عن المتطلبات Negotiating Requirements

- أنشطة التفاوض تتضمن تعريف الشركاء الأساسيين فى النظام، وتحديد شروط فوز الشركاء، يكون التفاوض للتصالح على تحويل شروط الفوز إلى حالة الفوز المتبادل لكل الاطراف بما فيهم المطورون والزبون.

13 تصحيح المتطلبات Validating Requirements

- يهتم بإظهار وإثبات أن المتطلبات تقوم فعلا بتعريف النظام الذى يريده المستهلك، تكلفة خطأ المتطلبات باهظة لذلك فإن التحقق منها له أهمية عالية، أن تصحيح خطأ المتطلبات بعد تسليم النظام قد يكلف ما يزيد بمائة مرة عن تكلفة تصحيح خطأ التنفيذ.

13-1- فحص واختبار المتطلبات Requirements checking

- **التحقق:** هل يوفر النظام الوظائف التى تلبى بأفضل أسلوب حاجات الزبون (المستهلك).
- **التماسك أو المتانة:** هل تتعارض المتطلبات مع بعضها.
- **الاكتمال:** هل تم تضمين كل الوظائف المطلوبة للمستهلك.
- **الواقعية:** هل يمكن تنفيذ كل المتطلبات فى حدود الميزانية المتاحة والتقنية المتوفرة.
- **قابلية التحقق:** هل يمكن فحص واختبار المتطلبات.

13-2- مراجعة المتطلبات Requirements review

مراجعات عامة يجب أن تنفذ خلال استنباط وصياغة تعريف المتطلبات، ويجب أن يقوم بهذه المراجعات كل من المستهلك وطاقم تنفيذ العقد ويمكن أن تكون المراجعات مكتوبة فى مستندات كاملة أو شفوية، ويمكن للاتصالات الجيدة بين المستهلكين ومطوري النظم أن تحل المشاكل فى مراحل مبكرة من العمل.

14-جودة مواصفات المتطلبات (SRS) Qualities of a Software Requirements Specification

1- ان تكون المتطلبات صحيحة Correct
2- ان تكون المتطلبات كاملة Complete
3- ان تكون المتطلبات رصينة Consistent
4- ان تكون المتطلبات غير غامضة Unambiguous
5- ان تكون المتطلبات مرتبة حسب الاهمية والاولوية Ranked for importance and stability
6- ان تكون المتطلبات قابلة للتحقق Verifiable
7- ان تكون المتطلبات قابلة للتحديث Modifiable
8- ان تكون المتطلبات قابلة للتتبع Traceable
9- ان تكون المتطلبات قابلة للفهم Understandable
10- ان تكون المتطلبات واقعية Executable

اذا تحققت كل الخطوات اعلاه يتم توثيقها بوثائقها الرسمية وتوزيعها على فرق عمل التطوير والمستفيدين من النظام ليستفاد منها في ازالة الغموض وتوضيح اعمال التصميم وبقية خطوات التطوير. حيث يتم الرجوع اليها عند اي لبس او اختلاف حول تطوير المشروع.

الفصل الرابع

نمذجة المتطلبات

REQUIREMENTS Modelling

ان كتابة الكلمات والسطور بصيغ متقنة مهمة جدا لتوضيح متطلبات المشروع وهذه تعتبر عجلة التواصل بين المهتمين بهذا العمل البرمجي. ولكن هذه الطريقة الكتابية ليست هي المثلى لتوضيح المتطلبات لمهندسي البرمجيات . بل ان نمذجة المتطلبات يجمع الصيغ المكتوبة والاشكال المعبرة عنها لتوضح المتطلبات بمفهومها الحقيقي المتجانس الغير قابل للتفسيرات والتأويلات من قبل الاطراف المعنية والمهتمة بتطوير المشروع البرمجي ، بالإضافة الى سهولة مراجعة خطواته وتصحيح اخطاءه وتكملة نواقصه المفقودة بين السطور الكتابية.

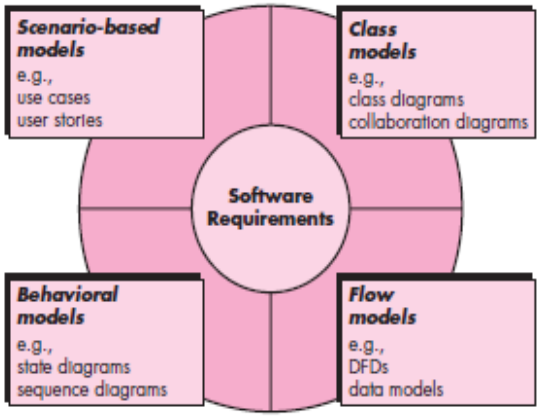
1-4- عملية نمذجة المتطلبات Requirements Modeling Process

- لتوضح ماذا يحتاج الزبون.
- لتحديد الاساس في انشاء تصميم البرمجية.
- لتعريف المتطلبات التي تكون محققة عند بناء البرمجية.

عندما نقوم ببناء نموذج سنحصل على فهم أفضل حول الكيان الواقعي ليتم بناؤه ,عندما يكون الكيان ملموس نستطيع بناء موديل مشابه له بالشكل والصيغة ولكنه أصغر بالحجم. أما بالنسبة للكيان غير الملموس (البرمجيات) فيجب أن يأخذ الموديل صيغة أخرى. إن الموديل يركز حول ما على النظام أن يعمل لا عن كيفية قيامه بذلك. ممكن أن يكون

الموديل على شكل رسومي وله بعض الأجزاء النصِّية لوصف المتطلبات. إن بناء موديل خلال مرحلة تحليل المتطلبات يخدم ادوار عديدة مهمة :

- إن النموذج يساعد المحلل على فهم معلومات ووظائف وسلوك النظام .
- ان النموذج يجب ان يركز على إظهار المتطلبات مع مجال المشكلة طور التحليل .
- إن النموذج يساعد كنقطة مركزية لمرحلة مراجعة المحددات Specification .
- إن النموذج هو أساس التصميم , حيث يجهز المصمم بالتمثيلات الأساسية الخاصة بالبرامجية .



مهندسو البرمجيات يقومون ببناء نماذج مستخدمين المتطلبات التي صيغت من قبل الزبون(العميل) هذه النماذج الرسومية تستخدم لتحقيق المتطلبات والتي نحتاج الى اختبارها من خلال اربعة نماذج هم – نماذج تعتمد المشهد - ونماذج تعتمد البيانات - نماذج تعتمد الاصناف والكائنات ونماذج تعتمد السلوك وكلاهم يمثل المتطلبات من كل الاتجاهات المختلفة ، وهذه الاعمال تزيد احتمال وجود الاخطاء بالمحتويات وطفوها الى السطح والتي لم تكن مكتشفة من قبل. وتكون الخطوات كالتالي:

1- نماذج –معتمدة - المشهد Scenario-based model

تمثل متطلبات النظام المختلفة بنقاط حالات الاستخدام ومشاهد المستخدم (الممثل)

عناصر وظيفية –هي عناصر قريبة من تسلسل المعالجات ووظائف النظام كاملة .

حالة الاستخدام – هي وصف لحالات التفاعل بين الممثل والنظام .

في هذه المرحلة تكتمل الرؤيا لكل اجزاء النظام (Use Cases) والادوار التي قام بها الممثلون (Actors) وكل الخطوات والسيناريوهات(Scenarios) وبالتالي قد يمثل النظام بنموذج حالة الاستخدام يمثله كاملا كما سنرى النماذج لاحقا:

2- نماذج الانتقال الموجه Flow-oriented models

نماذج متجهة الانسياب هي التي تمثل العناصر الوظائفية للنظام وكيفية انتقال البيانات وكأنها تتحرك خلال النظام.
مخططات انسياب البيانات Data- Flow Diagrams (DFD) ، ومخطط العلاقات والكينونات (ERD) Entity Relationship Diagrams

3-نماذج الكائنات الموجهة Class-Oriented models

نماذج الاصناف الموجهة تمثل الكائنات الموجهة لتلك الاصناف(خصائص ، عمليات) ، والاسلوب بحيث الاصناف تتعاون فيما بينها لإحراز متطلبات النظام.

هي تلك النماذج المعتمدة على الاصناف لتمثل الكائنات التي سيعالجها النظام. كل الاصناف لها اسماء يعرفها عن بعضها مأخوذة من طبيعة عمل او ما يمثله الصنف صراحة في النظام. وكل صنف له خصائصه المعطاة له من طبيعة عمله او اكتسبها او ورثها من اصناف اخرى بحسب موقعة ووظيفته المسنودة اليه. العمليات او (ما تسمى

ايضا بالطرق او الخدمات) التي سوف تطبق على الكائنات بتأثير المعالجة لها. العلاقات هي (بعض التركيبات المهيكلة) بين الكائنات والتعاون الحاصل وبين الاصناف المعرفة لها.

عناصر نماذج الاصناف تتضمن الكائنات والخواص والعمليات،ثم نماذج تقرير الصنف ، مخططات التعاون ثم اخيرا التحزيم (Packages)

نماذج السلوك Behavioural models

نماذج السلوك التي تظهر سلوك النظام بشكل متسلسل للأحداث الخارجية.

مخطط حالات الانتقال ومخطط حالات التسلسل.

طرق النمذجة Modeling Approaches

A. الطريقة الاولى: نمذجة مهيكلة Structured Modelling ,

وتستخدم لنمذجة المتطلبات (Requirements Modelling): تستخدم للتحليل المهيكل (Structure Analysis) والتي تعتبر البيانات والمعالجات هي التي تنقل البيانات كالكينونات (Entities) منفصلة. ثم كائنات البيانات تنمذج بطريقة تعرف خصائصها(Attributes) وعلاقاتها(Relations). المعالجات التي تعالج كائنات البيانات ونمذجتها بأسلوب يوضح كيفية انتقال البيانات كبيانات موجهة تدفق خلال النظام.

B. الطريقة الثانية: نمذجة الكائنات الموجهة Object Oriented Modelling

وتستخدم لنمذجة التحليل (Analysis Modeling): تستخدم لتحليل الاصناف المتجهة Object-Oriented (Analysis) ، هذه الطريقة تركز على تعريف الاصناف والاسلوب بحيث تتعاون فيما بينها البين حتى تؤثر على متطلبات الزبون.

النظام يتكون من عدة مخططات كل مخطط يصف جزء من تعقيداته ، وكل شكل يمتلك رموزه وخواصه ، وذلك كمايلي:

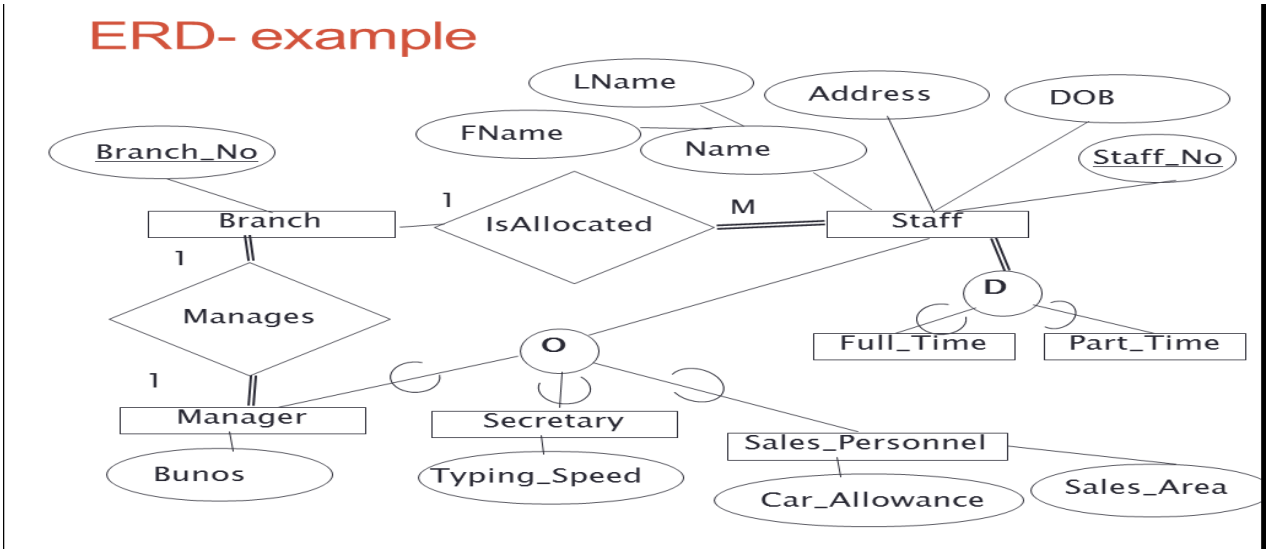
- وصف البيانات (Entity Relation Diagram -ERD) Data View
- وصف المعالجات Process View (Data Flow Diagram –DFD)
- وصف التحكم Control View (State Transition Diagram – STD)
- وصف واجهة المستخدم User Interface View (Graphical User Interface – GUI)
- وصف الشبكة والانترنت Internet View (Network Diagram)

وصف البيانات Data View

- تصف النظام من خلال حركة البيانات (Data) والمعطيات مع النظام (دخل - خرج) وداخل النظام نفسه.
- تغطي المخططات عمليا التحولات التي تجري على البيانات Data للنظام وتبين كيفية ترابط الـ Data في ما بينها ضمن اجزاء النظام ككل ولا تظهر المخططات المعالجة التي تتم عليها من خلال الـ Processes
- وتستخدم مخططات خاصة لوصف النظام من خلال ترابط الـ Data للنظام وهذه المخططات تدعى بمخططات Data Processing Diagram أو بما تدعى بشكل واسع بـ ERD.
- وهنا يتم توصيف النظام من خلال الرموز التالية :
- (الصفات Attribute - العلاقات Relationship – الكينونات Entity)

- دراسة ومعالجة البيانات للأنظمة أدى إلى نشوء فرع معلوماتي خاص يدعى Data Base Management System وهذه الانظمة تدعى اختصارا DBMS وهي تدرس وتعالج النظام من خلال تفاعلات الـ Data هذا النظام.

مثال على مخطط الكينونات ERD example



وصف المعالجات Processes V iew

- هنا يتم توصيف النظام من خلال الاعمال التي ينفذها هذا النظام وبالتالي إلقاء الضوء على الاعمال والاعمال الجزئية له وكيفية تأثيرها على سير عمل النظام
- ويتم إيجاد مخططات لاعمال النظام أو ما سوف ينفذه النظام ومن خلال هذه المخططات يفهم النظام بشكل أفضل من حيث الأعمال الرئيسية والاعمال الجزئية التي يجب عليه تنفيذها .
- تدعى المخططات بـ Data flow Diagram
- هنا يتم توصيف النظام من خلال الرموز التالية :(Process- Data Flow – Entity – Data Store)

وصف الانتقال Control View

- وهنا يتم إلقاء الضوء على العمل التحكمي في النظام من أجل معرفة التفاعلات والمؤثرات التحكمية الداخلية والخارجية له وتأثيرها على النظام .
- يُدرس النظام هنا من خلال دراسة الحالات التي يمر بها النظام من الحالة الابتدائية إلى الحالة النهائية ويتم توصيف حالات النظام بشكل دقيق والعبور للنظام من حالة إلى أخرى .
- وندرس هذه المخططات من خلال مخططات تدعى State Transition Diagram – STD
- ويتم توصيف النظام من خلال الرموز التالية : (State – Transition- event).

سوف يتم توضيحه لاحقا

وصف واجهة المستخدم User Interface View

- وجهة النظر هذه هي وجهة نظر تحليلية وتفاعلية مع المستخدم وتلقي الضوء على النظام من خلال الأوامر التحكمية من خلال التفاعل مع المستخدم .

- تعتبر عملية التحليل هذه بمثابة تصميم الواجهات التحكمية للنظام والتفاعلية مع المستخدم وتجعل المستخدم على إطلاع ومشاركة في التصميم من خلال الطريقة التفاعلية للنظام مع المستخدم عبر الواجهات التحكمية للنظام بشكل يجعل النظام أكثر مرونة في الاستخدام
- وهنا يستخدم فيها المخططات ذات الطابع التقليدي حسب كل نظام وهي الانظمة البرمجية التي تستخدم الشاشات والوجهات الخارجية (GUI) Graphical User Interface.

مثال:

وصف الشبكات Network View

- وهي عبارة عن شرح ترابط النظام مع الانظمة الاخرى من خلال الترابط الشبكي وطريقة هذا الترابط هي حديثة حيث ان الانظمة الحديثة اصبحت معقدة ومترابطة مع بعضها البعض ولا يوجد نظام مستقل بحد ذاته دائما هناك ربط للنظام مع الانظمة الاخرى
- كل وجه نظر من الوجهات السابقة تعطي عمليا فكرة عن عمله و كيفية عمل النظام و مجموع هذه وجهات النظر في بشكل أعم و أوسع لفهم النظام وبالتالي ويمكن القول ان الوجهات السابقة معا تقدم عمليا المتطلبات الحقيقية للنظام.
- وهناك فائدة أخرى لمخططات النظام وهي عندما يكون النظام معقد وهناك توزيع للأعمال على أشخاص متعددين بحيث تبقى المخططات هي سهلة الربط بين وجهات عاملة ضمن النظام والتي تسمح ايضا بتنفيذ المشاريع في مكاتب منفصلة أو بلدان مختلفة.

النمذجة المهيكلة The Structured Modelling

1. Data –flow diagram (DFD)
2. State transition Diagram (STD)
3. Combined Diagram (DFD+STD)

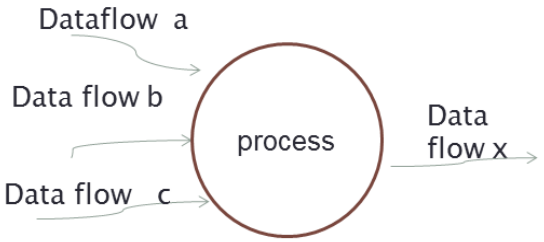
مخطط انسياب البيانات Data Flow Diagram

انسياب البيانات DFD يكون سهل الفهم ، ويمتلك اربعة ترميزات هي كمايلي:

- process .... المعالجة
- data flow انسياب البيانات
- data store مخزن البيانات
- entity ..... الكائن

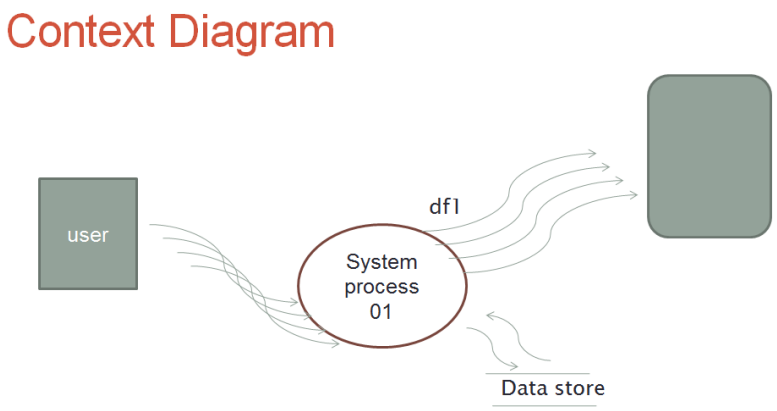
العملية Process: تعني العمليات التي سوف تجري على البيانات، وهي بمثابة أداء لتنفيذ دالة function برمجية لأحد لغات البرمجة حيث تقوم بعملية تحويل بيانات الدخل إلى بيانات أو معطيات خرج .

```
float function (int a, int b, int c)
{
```



```
….. ;
x=a+b+c;
return  x;
}
```

1- مخطط السياق Context Diagram



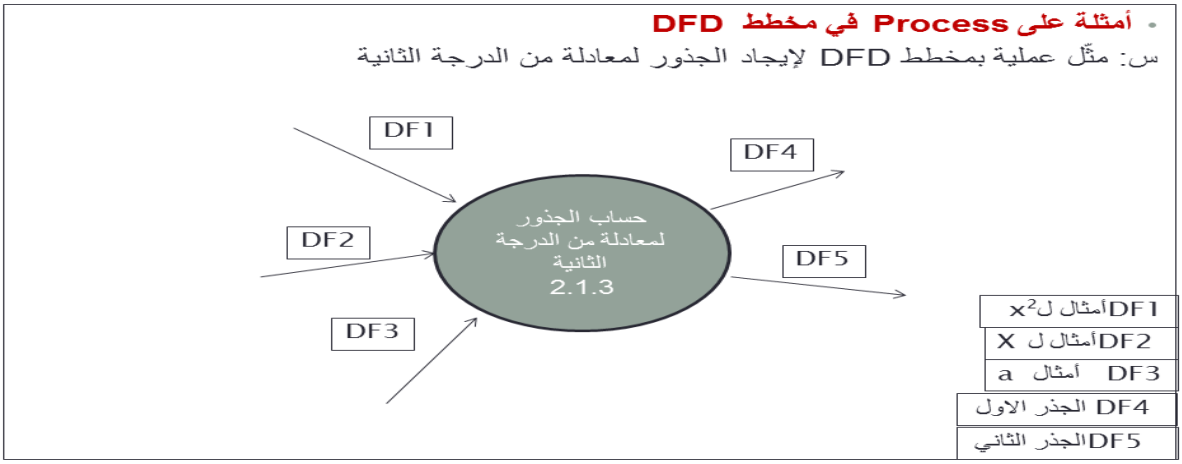
DFD Approach "tactic"

- استراتيجية مخطط انسياب البيانات DFD يكون من خلال رسم مخططات النظام على شكل مستويات كمايلي:
- قسم النظام الى نشاطات Divided the system into its activities
- قسم كل نشاك الى معالجات (عمليات) Divided the activity into processes
- قسم العملية الى اجزاء صغيرة مفهومة Divided the process into smaller till well understood

الشروط الواجب مراعاتها في مخططات DFD

1- يجب أن لا تكون جميع DF داخلة إلى Process
2- يجب أن لا تكون جميع DF خارجة من Process
3- يجب ان لا تكون مجموع الـ Processes في المستوي الواحد اكثر من 10 عمليات من أجل سهولة قراءتها
4- يجب أن يكون لكل DF شرح تفصيلي عنه يدعى بـ Date Dictionary في اسفل كل مخطط
5- كل DF داخل إلى DS يعني أن هنالك تسجيل في Data Store
6- كل DF خارج من DS يعني أن هنالك قراءة من Data Store
7- لكل Process في المستوي الأخير خوارزمية تعريفية بالـ Process تدعى بـ Process Specification 8- الـ Entities تكون في المستويات الاولى من المخططات
9- كل Process يرقم برقم خاص للمستوي – للعملية الرئيسية – العملية التفرعية
10- لا يوجد أي DF مباشر بين Entities
11- لا يوجد أي DF مباشر بين Data stores

امثلة: على عملية في مخطط انسيبا البيانات DFD in Process



أمثلة على التجزئة للـعملية Process in DFD

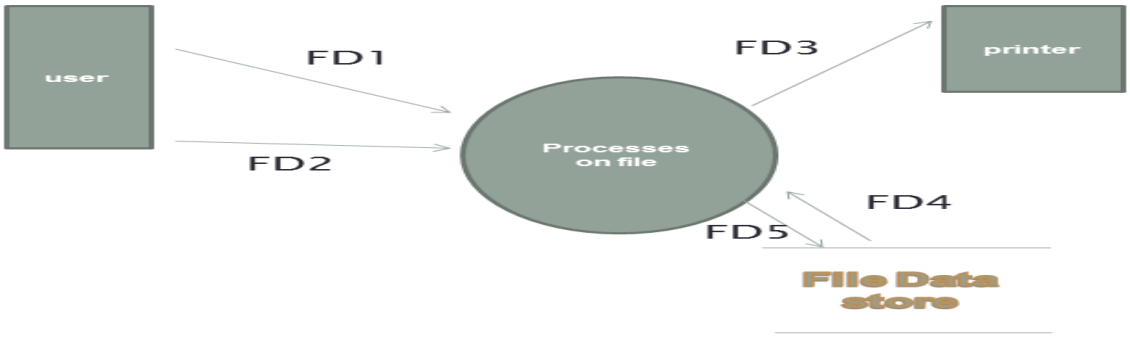
1- **في عمليات إدارة الملف هنالك عدة عمليات أهمها (إنشاء ملف جديد- فتح ملف – إغلاق ملف مفتوح – تخرين ملف – تخزين ملف تحت اسم ما – إنهاء ملف – طباعة ملف – إرسال ملف إلى - .....)**

2- يمكن لهذه العملية المعقدة أن تجزئ إلى عدة عمليات التي تجرى على الملف.

3- في المستوي الأعلى هنالك عملية تدعى بعملية إدارة الملف وفي المستوي اللاحق تجزئ هذه العملية إلى عمليات تفصيلية كما في الشكل التالي:

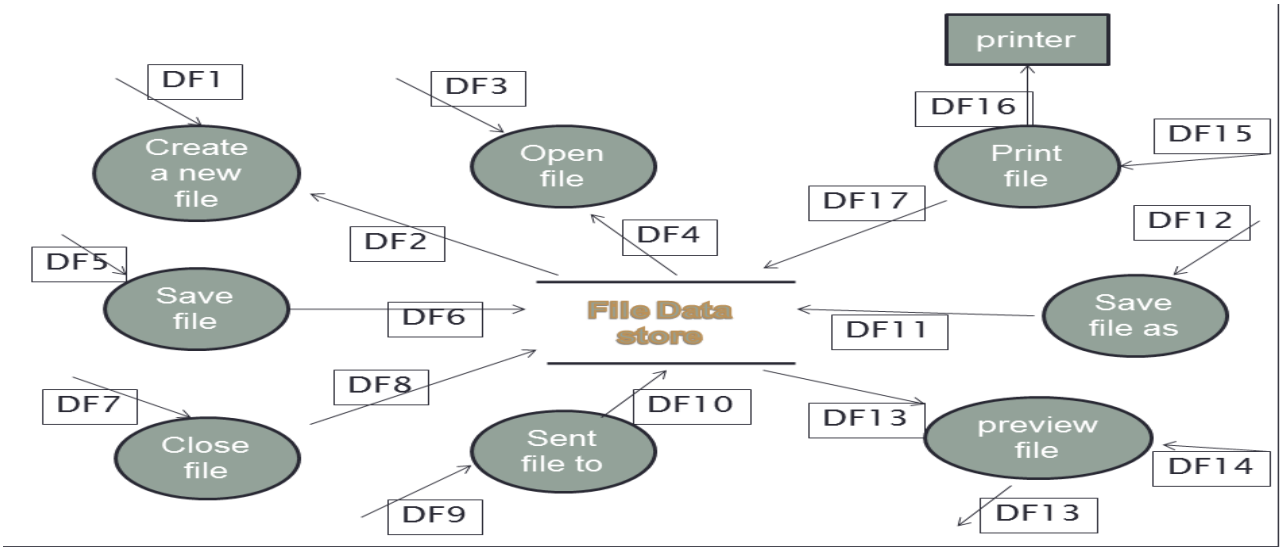
مثال: Process on File in DFD

في المستوي الاعلى (مستوى السياق Context Diagram) تمثل العمليات على الملف بعملية واحدة فقط ثم تجزئ في المستوي التالي إلى عدة عمليات.



شكل رقم( ):

تمثيل العمليات في المستوى الادنى



قسم مخزن البيانات الى الاصغر Divide Data Store in smaller

1- تستخدم DS واحد للنظام في المستويات العليا من مخططات DFD ، وخاصة في مخطط Context Diagram- والتي تعبر عن وحدة تخزين معطيات النظام ككل.

2- وفي المستويات الأدنى نحاول تجزئة الـ DS الكلية بحيث يتم إيجاد Data Entities تعبر عبر أجزاء تخزين مستقلة ذاتيا للحفاظ على ترابط بيانات النظام حسب مبادئ وأسس قواعد البيانات.

مثال: في نظام أتمتة أعمال سوبرماركت يمكن تجزئة الـ DS الكلية في مخطط النظام Context Diagram إلى DS1 خاصة بالسلعة وDS2خاصة بالمورد و DS3خاصة بطلبيات التوريد و DS4خاصة بالمخزن .

انسياب البيانات Data Flow

DF هو عبارة عن جملة بيانات أو معطيات النظام التي تمرر إلى عملية Processes وهنالك ثلاث أنواع لمصادرها :

1- من خارج النظام عبر Entities التي يمكن أن يمررها المستخدم مثلا

2- من مخزن البيانات DS للنظام والتي تعبر عن البيانات التي تكون مخزنة في النظام

3- من ناتج العملية Process والتي هي بيانات من عملية منفذة مسبقاً.

- **تمثيل انسياب البيانات DF في المستويات العليا (المستوى البيئي) تكون عمومية أي تكتب بشكل شمولي لإمكانية التعرف عليها فقط دون البوح عن التفصيل فيها**
- **أما في المستويات المتدنية فتكون بشكل دقيق ومعرفة بتفصيل والقيود التي يمكن ان تكون مشروطة بالإدخال مثلا أو عدد الحروف أو المجال العددي الذي يمكن قبول إدخاله أوما شابه ذلك .**

تطبيق عملي: لتحليل نظام سوبر ماركت Supermarket System Problem

اذا اردنا تحليل نظام سوبرماركت بسيط نقوم بتحديد الادارات:

- **الادارات في نظام السوبر ماركت هي:**

إدارة عمليات البيع – إدارة عمليات الطلبيات- ادارة عمليات تخزين السلع- ادارة عمليات بيانات الزبون - إدرة عمليات الجرد

ثم نحدد العمليات التي يقوم بها هذا النظام كمايلي:

- **العمليات التي تجرى على ادارة الطلبيات هي :**

انشاء طلبية جديدة - حذف طلبية – بحث عن طلبية – تعديل طلبية

- **وفي العمليات التي تجرى على ادارة البيع هي:**

تسجيل عملية بيع – حذف عملية بيع – تعديل عملية بيع – بحث عن عملية بيع

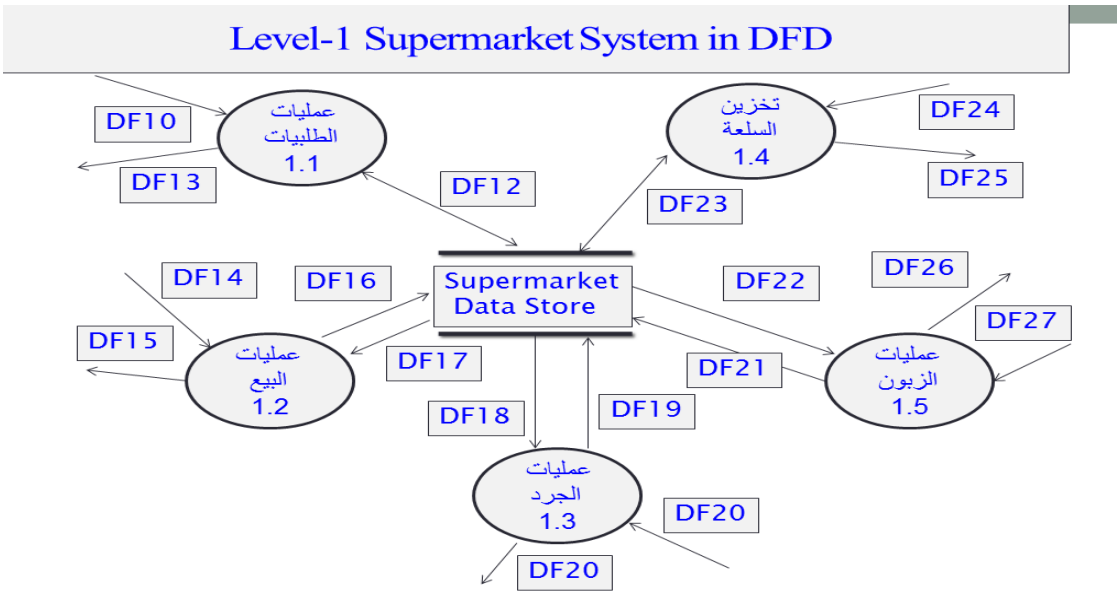
- **وفي العمليات التي تجرى على ادارة تخزين سلعة هي:**

إدخال سلعة جديدة –بحث عن سلعة -....

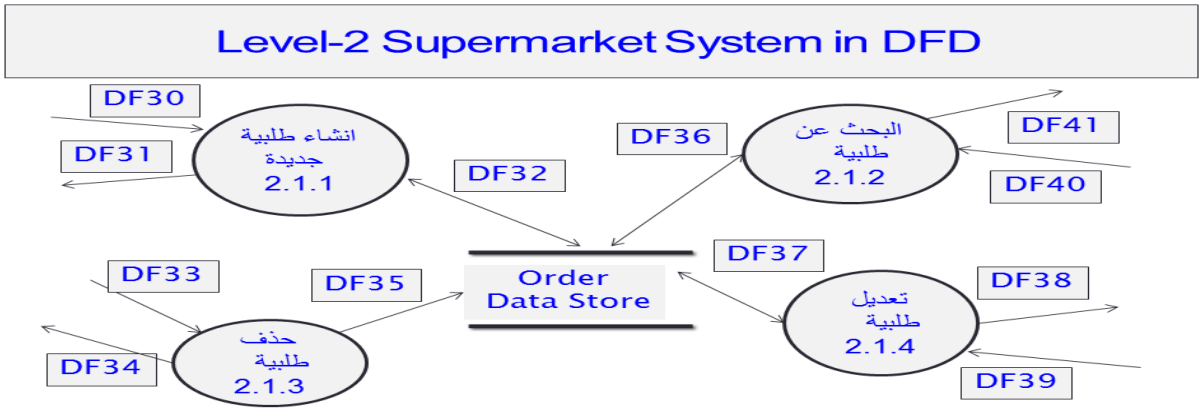
- **وفي العمليات التي تجرى على ادارة بيانات الزبون هي:**

يتم تعريف بزبون جديد - تعديل بيانات زبون –بحث عن زبون - حذف زبون .

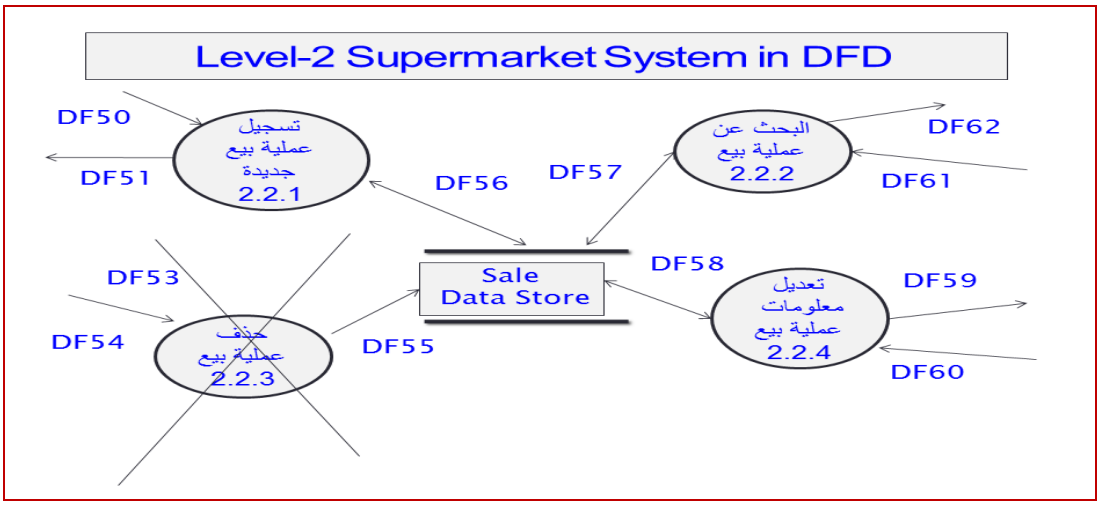
المستوى الاول لتمثيل العلميات في نظام لسوبر ماركت هي: عمليات البيع – عمليات الطلبيات- عمليات تخزين السلع- عمليات بيانات الزبون - عمليات الجرد



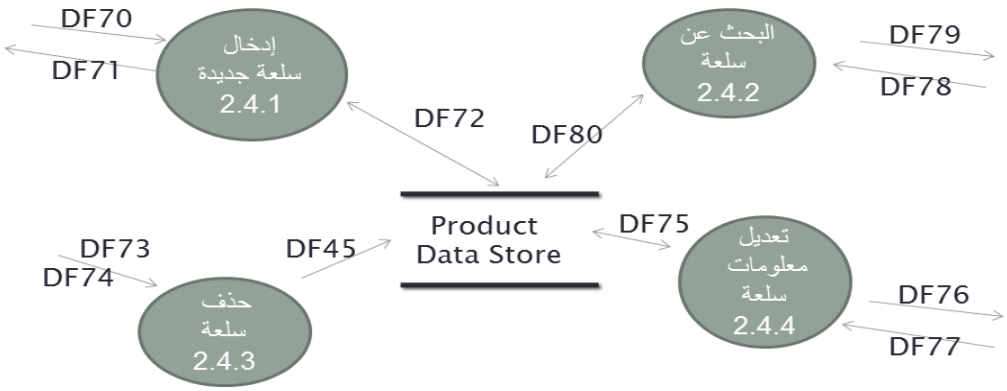
المستوى الثاني لتمثيل عمليات الطلبية هي : انشاء طلبية جديدة - حذف طلبية – بحث عن طلبية – تعديل طلبية



المستوى الثاني لتمثيل عمليات البيع هي: تسجيل عملية بيع – حذف عملية بيع – تعديل عملية بيع – بحث عن عملية بيع

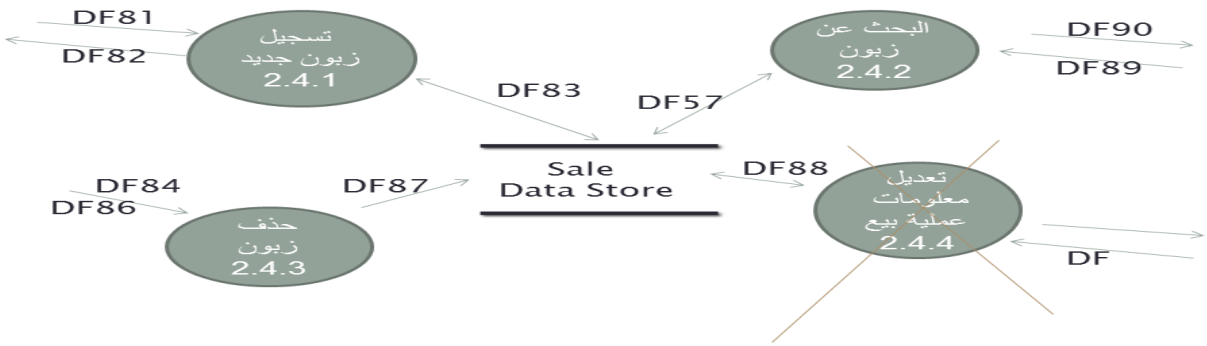


المستوى الثاني لتمثيل العمليات التي تجرى على ادراة تخزين سلعة هي: إدخال سلعة جديدة –بحث عن سلعة -....



المستوى الثاني لتمثيل العمليات التي تجرى على ادارة بيانات الزبون هي:

تسجيل زبون جديد - تعديل بيانات زبون –بحث عن زبون - حذف زبون .



المستوى الثاني لتمثيل عمليات البيع هي: تسجيل عملية بيع – حذف عملية بيع – تعديل عملية بيع – بحث عن عملية بيع

نمذجة النشاطات Activity Diagrams: تم شرحها مسبقا:

مخطط النشاط يوضح كيفية انسياب وظائف النظام. والذي يستخدم في نمذجة انسياب نشاطات اعمال المؤسسة، والذي يستخدم اثناء جمع المتطلبات لتوضيح كيفية انسياب الاحداث في نظام المؤسسة.

مخطط واقعة استخدام The Use Case Diagram

واقعة الاستخدام هي من أدوات UML القوية، هي ببساطة وصف لمجموعة من التفاعلات بين المستخدم و النظام. و من خلال بناء مجموعة من وقائع الاستخدام، يمكننا وصف كامل النظام الذي نخطط لإنشائه، بصورة واضحة و موجزة.

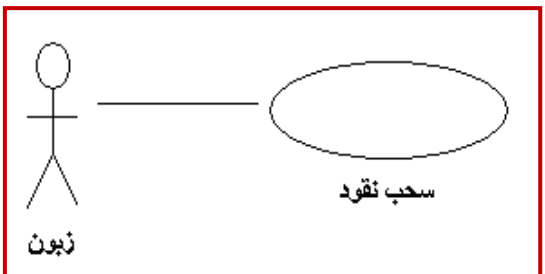
عادة ما يتم وصف وقائع الاستخدام باستعمال توليفات من (الفعل / الاسم) - على سبيل المثال: "دفع الفواتير" ، "تحديث المرتبات" أو "إنشاء حساب".

مثلا، إذا كنا نكتب برنامجا لنظام التحكم بالصواريخ، فإن وقائع الاستخدام المعتادة قد تكون: "إطلاق الصاروخ"، أو "بدء العدّ التنازلي".

بجانب الاسم الذي سنعطيه لواقعة الاستخدام، سوف نقدم شرحا نصيا كاملا للتفاعلات التي ستنشأ بين المستخدم و النظام. هذه الشروح النصية سوف تنتهي في الغالب لتكون أكثر تعقيدا، لكن UML تقدم لنا ترميزا بسيطا و مدهشا لتمثيل واقعة الاستخدام، كالتالي:

اللاعب Actor

واقعة الاستخدام لا يمكنها بدء الأحداث أو التفاعلات من تلقاء نفسها. **اللاعب** هو شخص ما الذي **يمكنه** بدء أو تفعيل واقعة الاستخدام. مثلا، إذا كنا نقوم بتطوير نظام مصرفي، و كان لدينا واقعة استخدام تسمى **"سحب النقود"** ، فيمكننا الإقرار بأننا نحتاج **لزبائن** للتمكن من سحب هذه النقود، على ذلك سيكون **الزبون** أحد اللاعبين لدينا. مرة أخرى، الترميز لهذا اللاعب سيكون بسيطا:



علاقة لاعب بواقعة استخدام

لو تعمقنا أكثر، فإن اللاعبين يمكن أن يكونوا أكثر من مجرد أناس. اللاعب قد يكون أي شيء خارج النظام يقوم بتفعيل واقعة الاستخدام، مثل جهاز حاسوب آخر. أكثر من ذلك قد يكون اللاعب مفهوما أكثر تجريدا مثل **الوقت**، أو **تاريخا** معينا.

مثلا، قد يكون لدينا واقعة استخدام اسمها **"حذف الطلبيات القديمة"** في منظومة لمناولة الطلبيات، و اللاعب الذي سيقوم بتفعيل هذه الواقعة قد يكون تاريخ **"آخر يوم عمل"**. كما لاحظنا، اللاعبون مرتبطون بوقائع الاستخدام،

حيث أن اللاعب هو الذي سيقوم بتفعيل أو بدء واقعة استخدام معينة. يمكننا تمثيل ذلك عل مخطط واقعة استخدام من خلال وصل اللاعب بواقعة الاستخدام:

من الواضح أنه بالنسبة لمعظم الأنظمة، يمكن للاعب الواحد التفاعل مع مجموعة من وقائع الاستخدام، كما أن واقعة الاستخدام الواحدة يمكن تفعيلها من قبل أكثر من لاعب مختلف. هذا يقودنا إلى مخطط واقعة استخدام متكامل، كما سنرى.

الغرض من وقائع الاستخدام

إن بساطة تعريف **"واقعة الاستخدام"** او **"اللاعب"**، مع بساطة تخيّل وقائع الاستخدام من خلال نموذج UML، سوف تجعلنا معذورين إذا اعتقدنا أن وقائع الاستخدام أمرها هيّن و سهل، و أنها أبسط من أن نقلق بشأنها. هذا خطأ. إن وقائع الاستخدام قوية بصورة كبيرة.

- وقائع الاستخدام تحدد النطاق العام للنظام. إنها تمكننا من تصوّر حجم ونطاق عملية التطوير بالكامل.
- وقائع الاستخدام شبيهة جدا بالمتطلبات، و لكن بينما المتطلبات تميل لأن تكون مبهمة و مربكة و ملتبسة و مكتوبة بشكل سيء، نجد أن البناء المحكم لوقائع الاستخدام يجعلها أكثر تركيزا.
- مجموع وقائع الاستخدام تشكل النظام بالكامل. مما يعني أن أي شيء لم يتم تغطيته في وقائع الاستخدام هو خارج حدود النظام المراد تطويره. لذا فإن مخطط وقائع الاستخدام متكامل بدون فجوات.
- إنها تمكن من التواصل بين العملاء و المطورين (ما دام المخطط بهذه السهولة، فالكل يستطيع فهمه).
- وقائع الاستخدام ترشد فرق التطوير خلال عمليات البناء التطوير- سوف نرى أن وقائع الاستخدام بمثابة العمود الفقري لعمليات التطوير، و ستكون المرجع لنا في كل ما نصنعه.
- سوف نرى كيف أن وقائع الاستخدام تقدم منهجية لتخطيط عملنا في التطوير ، و تسمح لنا بتقدير الوقت اللازم لإنجاز العمل.
- وقائع الاستخدام تقدم الأساس لبناء اختبارات النظام .
- أخيرا، فإن وقائع الاستخدام تساعد في إعداد أدلة التشغيل.

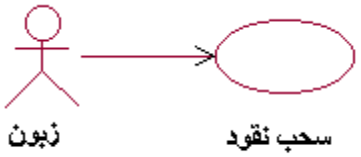
غالبا ما تصدر ادعاءات بأن وقائع الاستخدام هي ببساطة أسلوب للتعبير عن متطلبات النظام. واضح أن كل من يقول بهذا قد غاب عنه الغاية من وقائع الاستخدام!

مخطط واقعة استخدام مفيد؟

واقعة الاستخدام يجب أن تحقق هدفا ينشده اللاعب

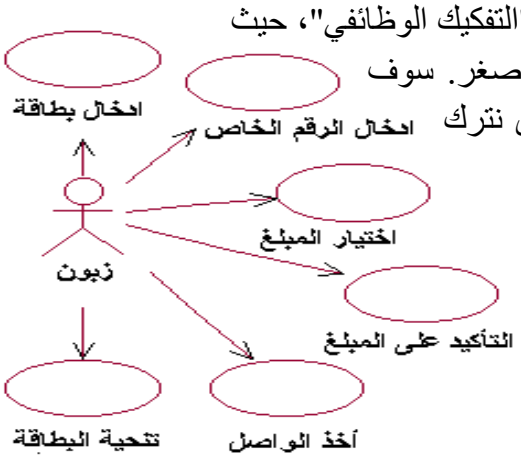
بتطبيق هذه القاعدة البسيطة على مثالنا الصراف الالي، يمكن أن نطرح سؤالا: **"هل الحصول على الواصل"** هو هدف الزبون؟ حسنا، ليس تماما. سوف لن ينتهي العالم إذا لم يتم إصدار هذا الواصل.

بتطبيق القاعدة على وقائع الاستخدام الأخرى، سوف نجد أنه في الحقيقة لا أحد منها تصف الهدف الذي ينشده المستخدم. هدف المستخدم هو **سحب النقود**، وهذا ما يجب أن تكون عليه واقعة الاستخدام!



واقعة استخدام أكثر تركيزا

هذه المقاربة قد تكون مؤلمة في البداية، حيث أننا قد تعودنا على "التفكيك الوظائفي"، حيث يتم تحليل و كسر المهام المعقدة و تحويلها إلى مهام أصغر و أصغر. سوف نرى لاحقا أن وقائع الاستخدام يمكن تفكيكها، لكننا الآن يجب أن نترك هذه الخطوة إلى الوقت الذي نبدأ فيه بعملية البناء.



حالات الاستخدام لنظام الصراف الالي:

- **إدخال البطاقة** **-الزبون يدخل البطاقة**
- **إدخال الرقم الخاص** **-الزبون يدخل الرقم الخاص**
- **اختيار المبلغ المطلوب** **- الزبون يختار المبلغ**
- **التأكيد على المبلغ المطلوب** **- الصراف يتأكد من المبلغ**
- **تنحية البطاقة** **-الصراف يخرج البطاقة**
- **اخراج المبلغ** **- الصراف يخرج المبلغ**
- **أخذ الواصل** **-الزبون يأخذ المبلغ**

مخطط واقعة استخدام مفيد؟

توصيفات وقائع الاستخدام

كل واقعة استخدام تحوي مجموعة كاملة من التفاصيل النصية عن التفاعلات و التصورات التي تشملها الواقعة.

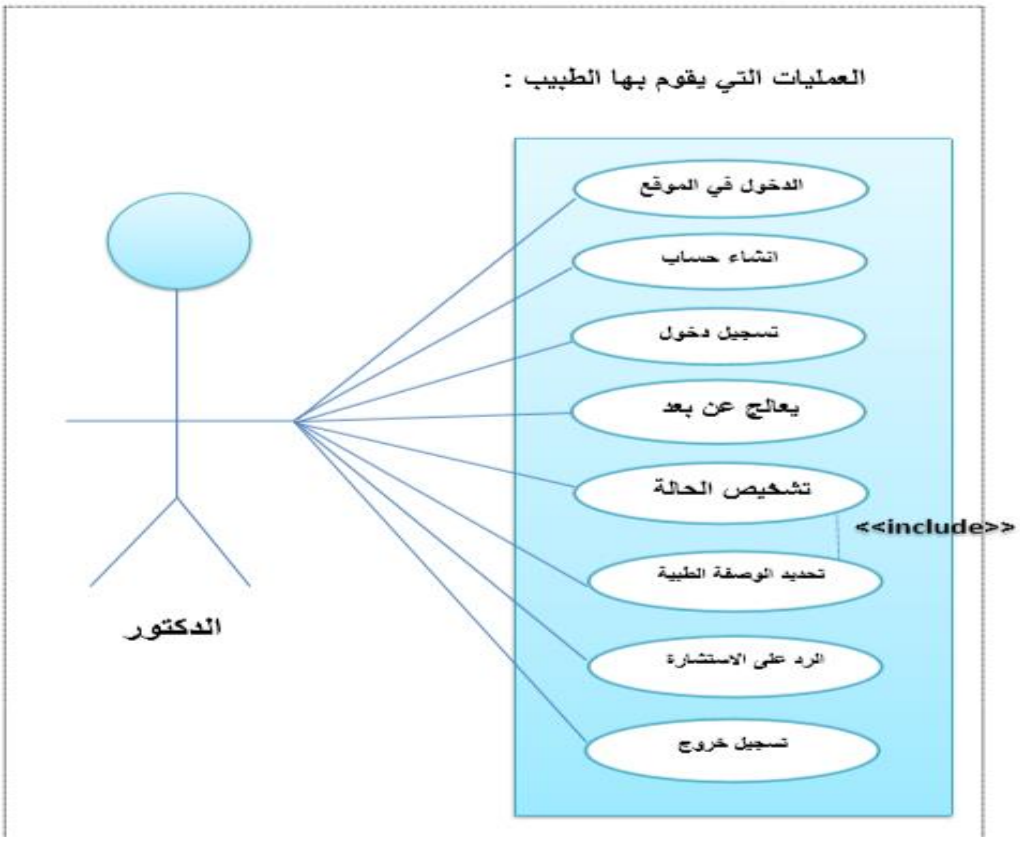
نلاحظ أنUML لا تحدد ما يجب أن يكون عليه شكل أو محتويات هذه الوثيقة - هذا يرجع للمشروعات أو الشركات لتحدده كيفما يناسبها[1]. بالنسبة لنا سوف تستعمل النموذج التالي:

واقعة الاستخدام	اسم واقعة الاستخدام
وصف مختصر	وصف موجز لواقعة الاستخدام
شروط سابقة	وصف للشروط التي يجب أن تتوفر قبل تفعيل واقعة الاستخدام
شروط لاحقة	وصف لما سيحدث عند انتهاء واقعة الاستخدام
المجريات الأساسية	قائمة بتفاعلات النظام التي ستأخذ مكانها وفق أكثر التصورات شيوعا. مثلا، بالنسبة لواقعة "سحب النقود"، ستكون "إدخال البطاقة"، "إدخال الرقم الخاص"، و هكذا ..
مجريات بديلة	وصف للتفاعلات البديلة المحتملة.
مجريات استثنائية	و صف للتصورات المحتملة عندما تقع أحداث غير متوقعة، أو لا يمكن التنبؤ بها.

نموذج لتوصيف واقعة استخدام

مثال: حالات الاستخدام لاحد وظائف نظام العيادة النفسية للمعالجة عن بعد؟

حالات الاستخدام التي تمثل العلميات التي يقوم بها الطبيب في هذا النظام:



توصيف عملية انشاء حساب

مواصفات حالة الاستخدام			
رقم العملية	٢	نسخة العملية	1.0
اسم العملية	انشاء حساب		
التاريخ		الاولوية	عالي

المنفذ :

- المستخدم

الخلاصة :

-انشاء حساب للمستخدم في الموقع

السيناريو :

- فتح الموقع
- عرض صفحة انشاء حساب
- اضغط على زر انشاء حساب
- كتابة الاسم بالكامل
- ادخل الايميل
- كتابة كلمة المرور
- اعادة كتابة كلمة المرور مرة اخرى
- اختر المنطقة الزمنية
- اضغط على زر متابعة

القيود :

- يجب تثبيت النظام
- يجب تجنب استخدام الرموز الخاصة

توصيف عملية تسجيل الدخول الى الموقع:

مواصفات حالة الاستخدام			
رقم العملية	٢	تسخة العملية	1.0
اسم العملية	تسجيل الدخول للمستخدم		
التاريخ		الاولوية	عالي

المنفذ :

- المستخدم

الخلاصة :

-تسجيل الدخول للمستخدم (المريض) في الموقع

السيناريو :

- فتح الموقع
- عرض الصفحة الرئيسية للموقع
- ادخل الايميل
- ادخل كلمة المرور
- اضغط على زر تسجيل

القيود :

- يجب تثبيت النظام
- يجب تجنب استخدام الرموز الخاصة

توصيف عملية الحجز الالكتروني:

مواصفات حالة الاستخدام			
رقم العملية	٣	تسخة العملية	1.0
اسم العملية	الحجز الالكتروني		
التاريخ		الاولوية	عالي

المنفذ :

- المستخدم

الخلاصة :

-حجز الالكتروني للمريض لاخذ موعد

السيناريو :

- فتح الموقع
- عرض الصفحة الرئيسية للموقع
- تسجيل الدخول
- فتح الصفحة الخاصة بالحجز
- قم بالاختيار باما الحجز الفعلي للعيادة او الحجز للعلاج عن بعد
- قم بتاكيد الحجز

القيود :

- يجب تثبيت النظام
- يجب الحجز لتاكيد الحجز والموقع من قبل العيادة

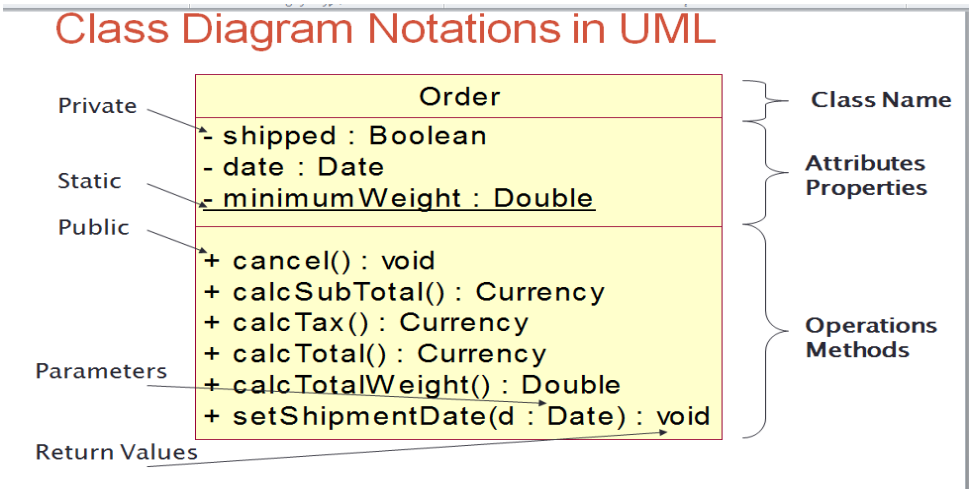
ملخص: وقائع الاستخدام أسلوب فعال لنمذجة ما يحتاج النظام لعمله.

هي طريقة ممتازة للتعبير عن نطاق عمل النظام (ما بالداخل = مجموع وقائع الاستخدام؛ ما بالخارج = اللاعبون). نحتاج إلى أن ننتبه لمدى كثافة وقائع الاستخدام التي تحوي تعقيدات. أفضل وسيلة لبناء وقائع الاستخدام هي مع الزبون في ورشة عمل.

نمذجة الكائنات الموجهة Object Oriented Modeling Methods

جميع الطرق تتحدث عن تحليل وتصميم للأنظمة من خلال رؤية للنظرية الحديثة (OO) وبالتالي إنشاء تصاميم للنظام بما يناسب متطلبات هذه النظرية ولكن مع إختلافات بسيطة وليست جوهرية ماهي متطلبات نظرية (OO) ؟

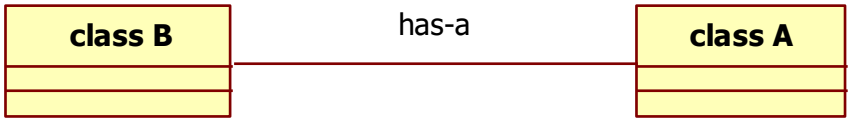
- نظرية (OO) تتحدث عن مفاهيم جديدة في فهم وتحليل وتصميم وتنفيذ الأنظمة البرمجية الحديثة من خلال مفهوم الـ OBJECT ومفهوم الـ CLASS ومتطلبات هذه المنهجية هي إنشاء مخططات متناسبة والمفاهيم الجديدة .



العلاقات بين الاصناف Classes Relationships

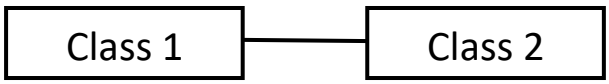
العلاقات الرئيسية بين Classes

- هنالك علاقات رئيسية بين الـ classes* وهي علاقة الـ has –a والتي تسمى compose or association بين الاصناف الـ classes.



التجمعية Association

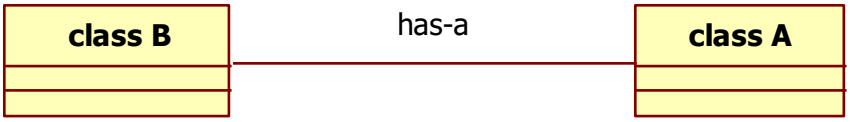
- **Represents links between classes.**
  - **Customer makes a payment.**
  - **Customer makes an order.**



- **UML allows non-directional associations at the analysis phase**

We have

- **ordinary association**

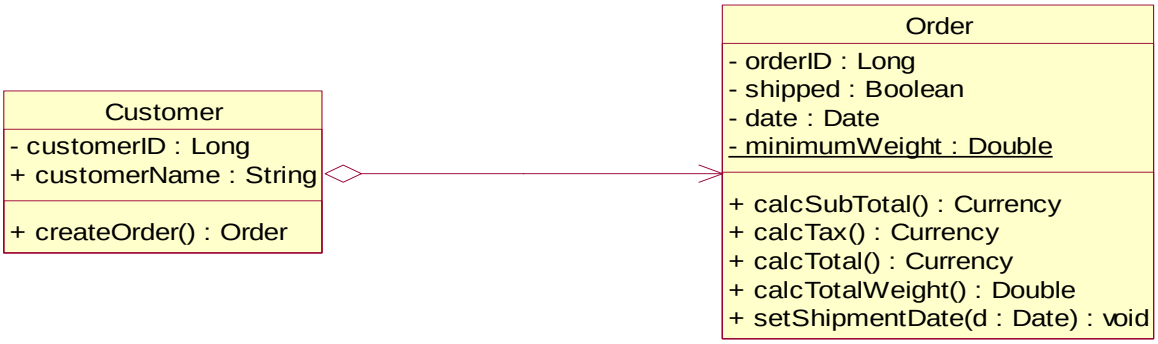


- **simple aggregation:**

very similar to ordinary association, only means one class is more Important than the other.

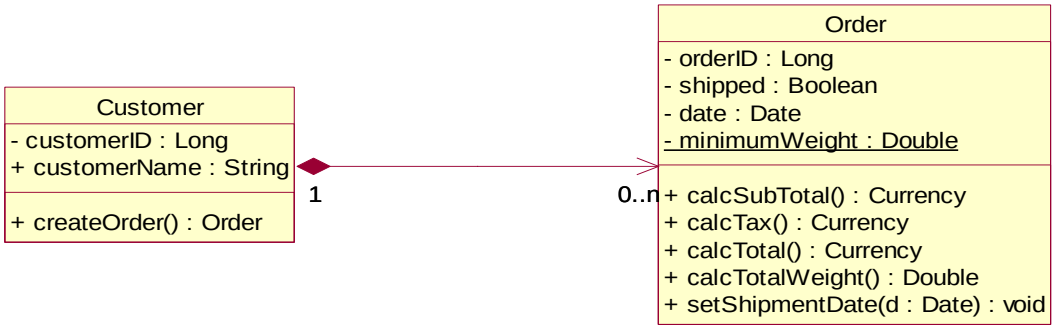
Aggregation is modeled by an empty diamond at the important part For the programmers, Aggregation is "Call by reference"

- **simple aggregation**



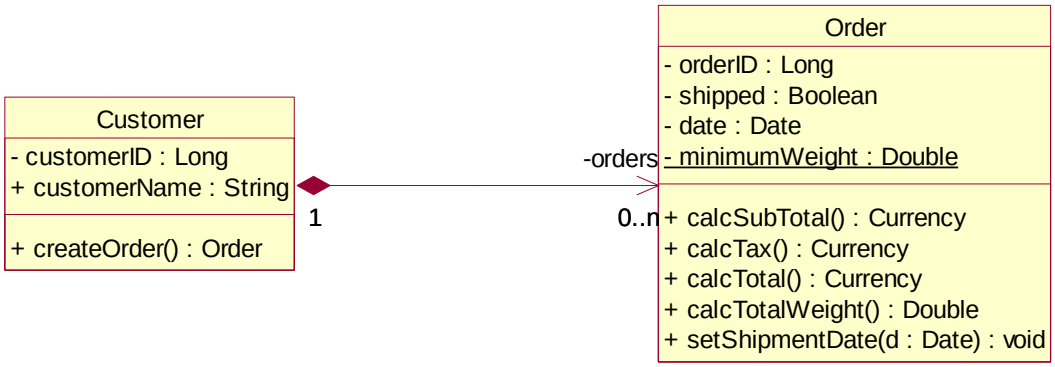
Associations have Multiplicity

- **0**
- **1-1**
- **1 .. n**
- **n….n**



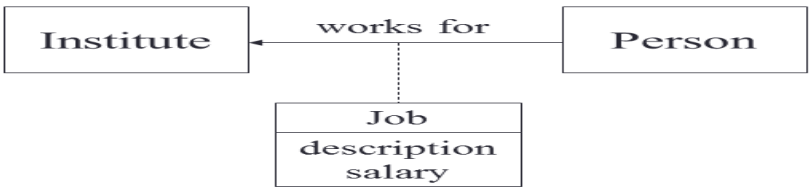
Associations have Roles

- Roles explain the meaning of the association
- Roles are “names” for the participation of each class in the association.
- The programmer converts the roles to the appropriate property names.
- Roles have visibility



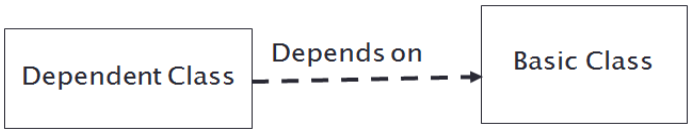
Association Classes

- **Associations can have properties and methods.**
- **An Association Class is a class that represents the association.**



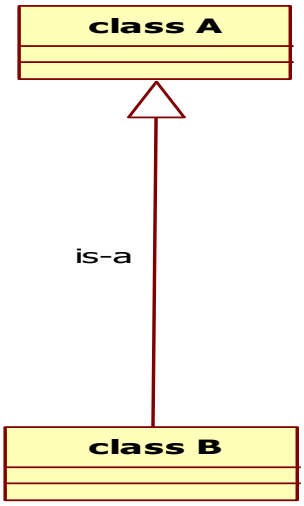
Dependency:

- **Dependency means “One class uses the other”**
- **A dependency relationship indicate that a change in one class may effect the dependent class, but not necessarily the reverse.**
- **You use dependency when you wants to indicate that one thing uses another.**
- **Often used to indicate that a method has object of a class as arguments.**

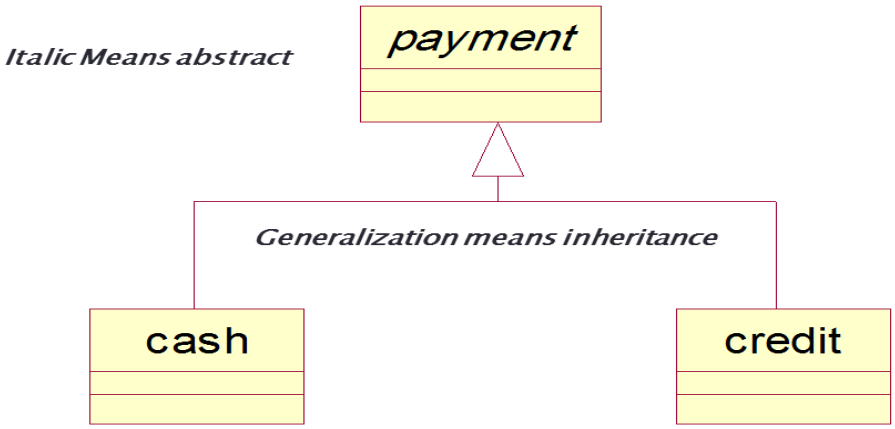


العلاقة الرئيسية الثانية هي علاقة التوارث Inheritance :

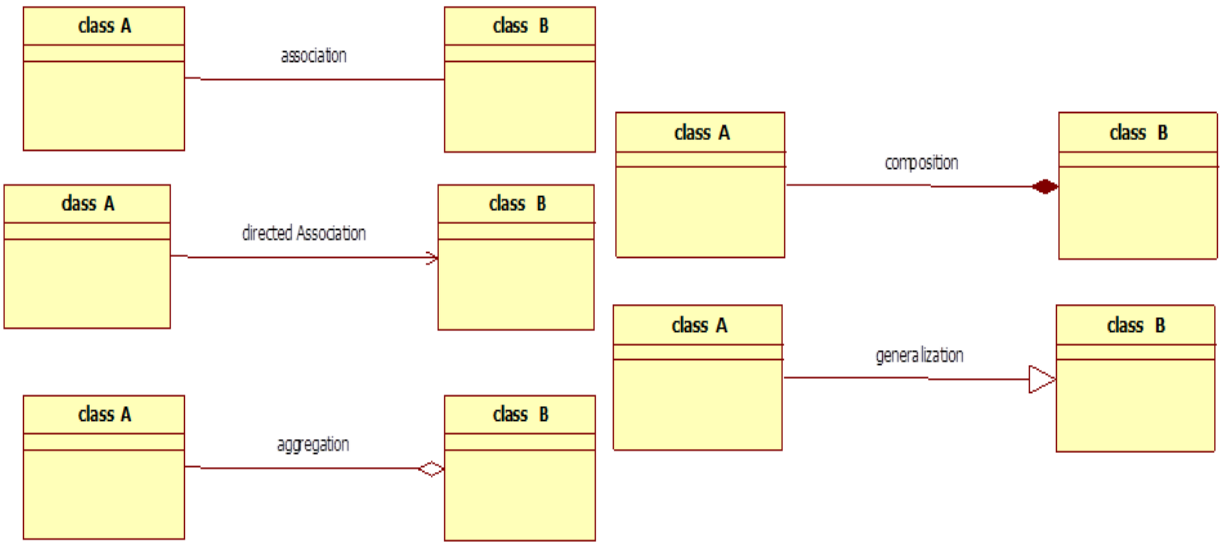
علاقة الـ is–a والتي تسمى generalization بين الاصناف الـ Classes



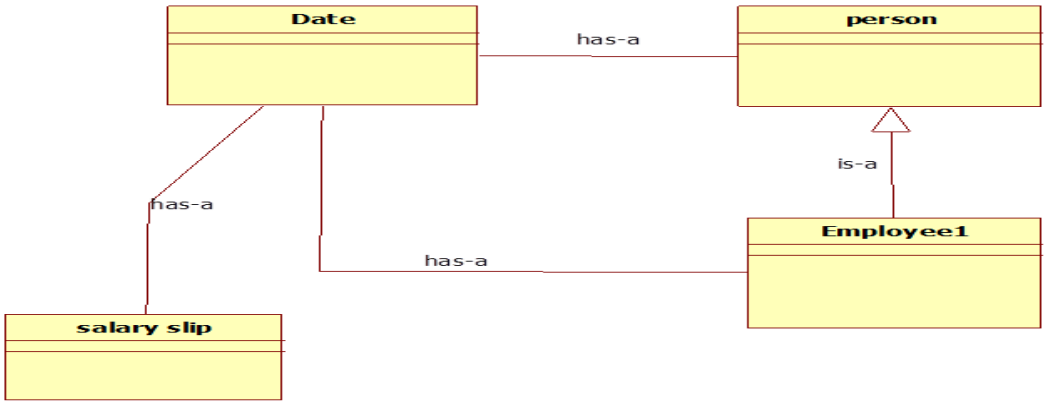
الاصناف المجردة Abstract Classes



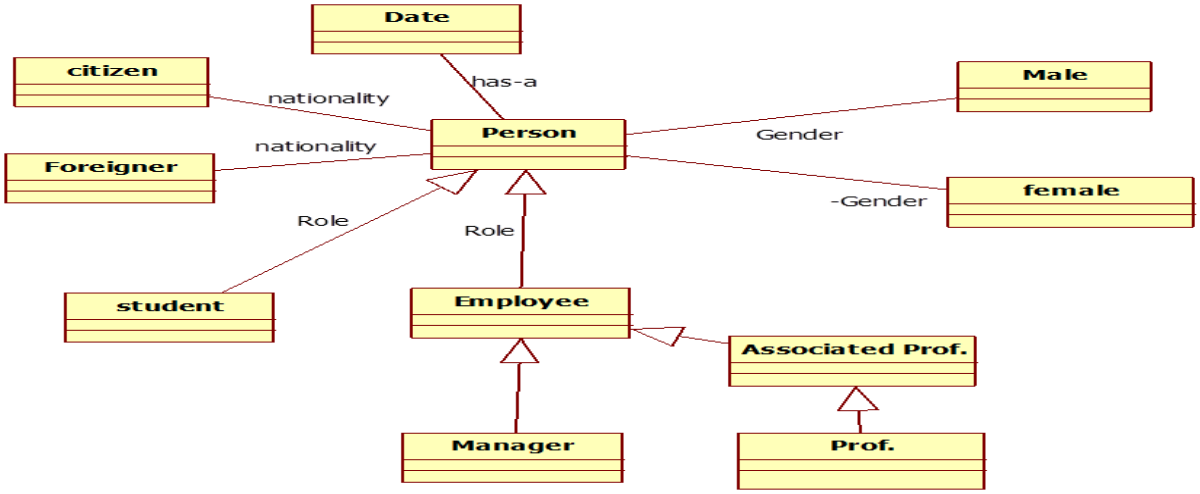
الرموز المستخدمة في class diagram: العلاقات بين الاصناف في لغة النمذجة الموحدة UML



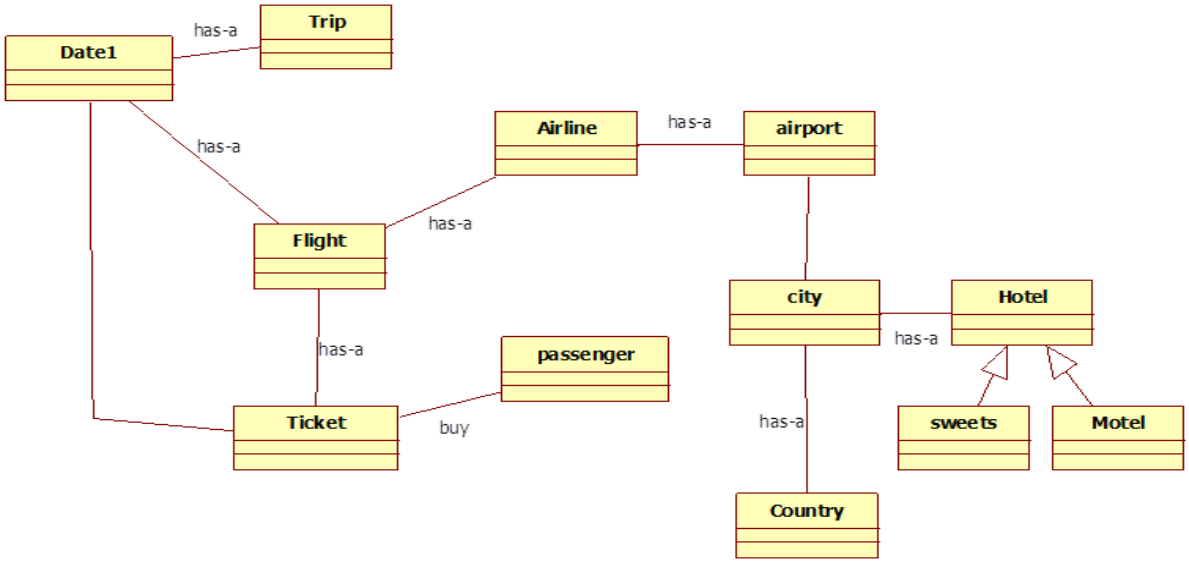
مثال بسيط لمخطط الاصناف



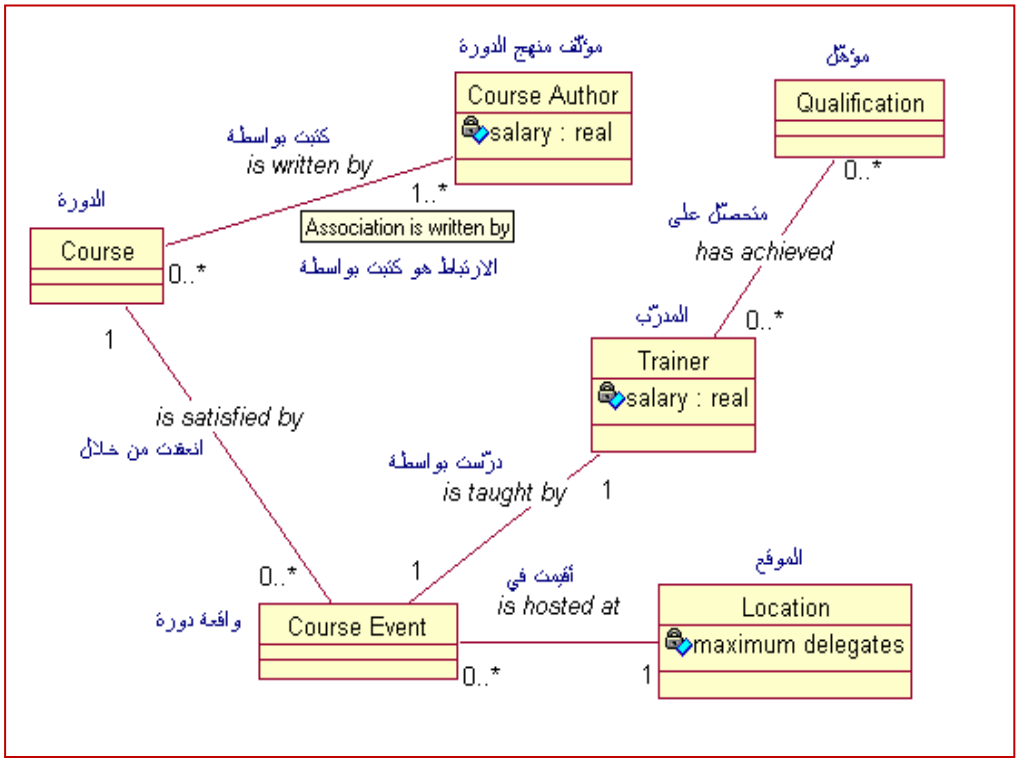
مثال اخر بسيط لمخطط الاصناف



مثال اخر بسيط لمخطط الاصناف



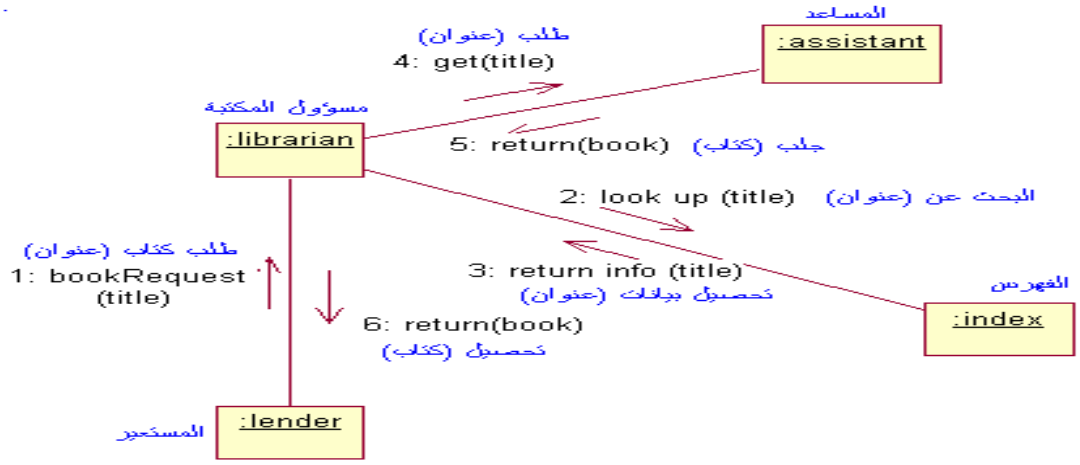
مخطط الاصناف The Class Diagram



رسم مخططات الاصناف جانب أساسي لأي منهج للتصميم بالمنحى للكائن، لذلك ليس بالغريب أن تقدّم لنا UML الصيغة المناسبة لها. سوف نرى أنه بإمكاننا استخدام مخطط الاصناف في مرحلة التحليل و كذلك في مرحلة التصميم - سوف نقوم باستعمال صيغ مخططات الاصناف لرسم خريطة للمفاهيم العامة التي يمكن للمستفيد (الزبون) أن يستوعبها (و سوف نسمّيها النموذج المفاهيمي (Conceptual Model). و هي بالإضافة إلى مخطط وقائع الاستخدام، تجعل من النموذج المفاهيمي أداة قوية لتحليل المتطلبات.

مخططات التعاون The Collaboration Diagrams

و نحن نقوم بتطوير برامج المنحى الكائني؛ إذا احتاج برنامجنا لأن يقوم بأي شيء فسيكون ذلك بواسطة

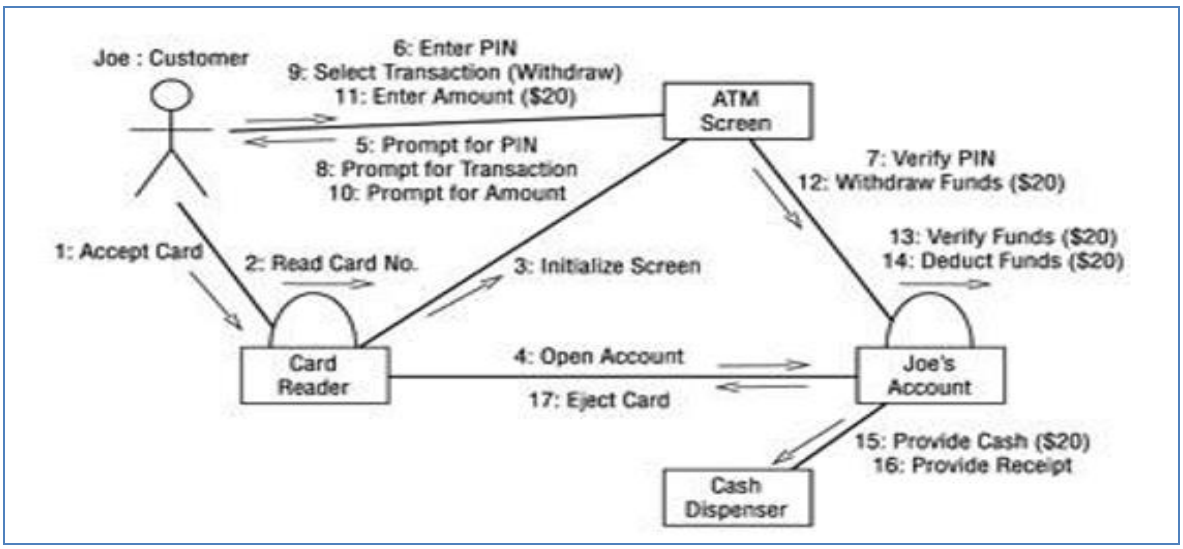


تعاون الكائنات. يمكننا رسم مخططات التعاون لوصف الكيفية التي تتعاون بها الكائنات فيما بينها بالطريقة التي نريدها.

هنا مثال جيد عن لماذا UML هي مجرد صيغة أكثر من كونها عملية حقيقية لتطوير البرمجيات. سوف نرى أن ترميز UML للمخطط بسيط جدا، و لكن عملية تصميم تعاون فعّال، (لنقل "تصميم برنامج راسخ و يسهل صيانته") ، يعدّ صعبا بالتأكيد. ربما علينا تخصيص فصل كامل يتناول الخطوط العريضة لمبادئ التصميم الجيّد، مع أن الكثير من مهارات التصميم تأتي من الخبرة.

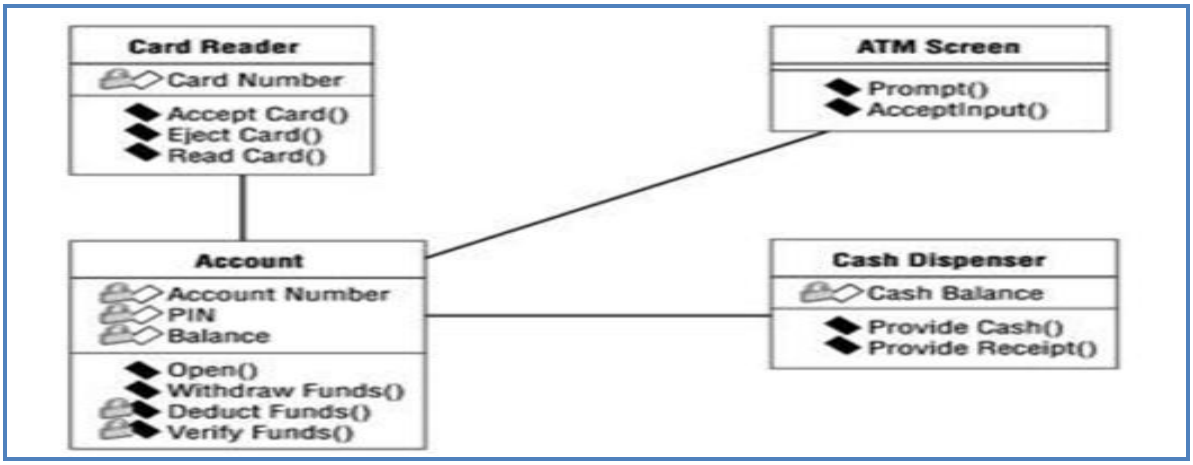
- **مخطط التعاون Collaboration Diagrams**

- Collaboration diagrams show exactly the same information as the Sequence diagrams. However, Collaboration diagrams show this information in a different way and with a different purpose. The Sequence diagram illustrated in Figure 1.11 is shown in Figure 1.12 as a Collaboration diagram.



- **مخططات الكائن Class Diagrams**

- Class diagrams show the interactions between classes in the system. Classes can be seen as the blueprint for objects, as we'll discussed earlier. Customer's account, for example, is an object. An account is a blueprint for Joe's checking account; an account is a class. Classes contain information and behavior that acts on that information. The Account class contains the customer's PIN and behavior to check the PIN. A class on a Class diagram is created for each type of object in a Sequence or Collaboration diagram. The Class diagram for the system's Withdraw Money use case is illustrated in Figure ().



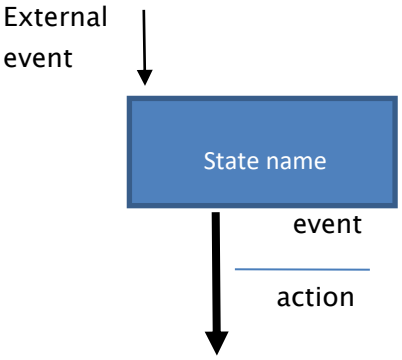
Class diagram for the ATM system's Withdraw Money Use Case

مخطط حالة الانتقال State transition Diagram (STD)

- في هذا النوع التحليلي للأنظمة فإنه يدرس فيه الجزء التحكمي للنظام حيث يلقي الضوء على النظام من وجه تحكمية من حيث تفاعل النظام مع المؤثرات الخارجية والداخلية بالإضافة إلى حالات إنتقال النظام من حالة مستقرة إلى حالة مستقرة أخرى ويتم رسم مخططات للنظام وفق للحالات التي يمر بها النظام.
- System State تمثل عملياً الحالة التي يتواجد بها النظام في حالة مستقرة فمثلاً يمكن القول عن نظام المصعد أن حالته في وضع الانتظار للطلب هي System State له وتمثل بالمخطط بمستطيل يحمل الاسم لهذه الحالة .
- يمكن تمثيل النظام بالحالة الابتدائية والحالة الوسيطة والحالة النهائية حيث أن الحالة الوسيطة تمثل حالات النظام كلها التي تقع بين الحالة الأولى والحالة النهائية .

مثال: رموز حالة الانتقال STD: حالة الانتقال System state : تمثل بمستطيل يحمل اسم هذه الحالة

- **System transition**



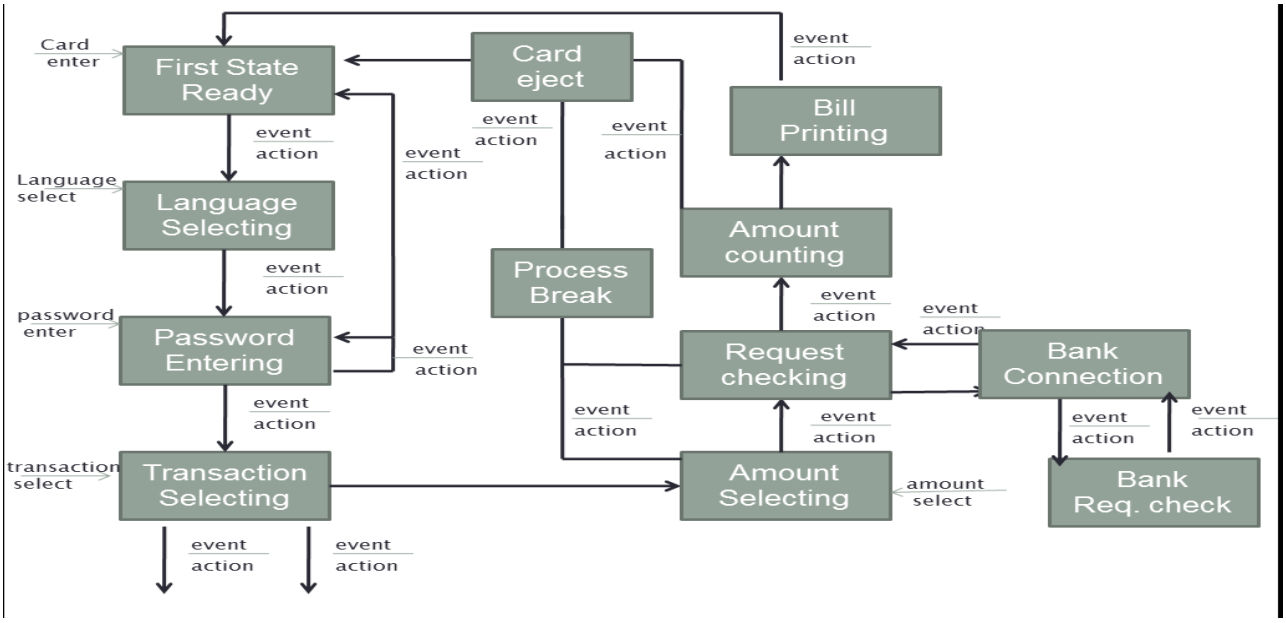
- **external event:** وهو المؤثر الخارجي الذي يؤثر على النظام ويسبب له تغيير في حالته والمؤثر الخارجي يمثل بسهم يتواجد بجوار مستطيل الحالة للنظام. وهنالك نوع من المؤثرات وهو المؤثر الداخلي الذي يؤثر بالنظام أيضا وينقل حالته من حالة مستقرة إلى حالة مستقرة أخرى
- **System transition** ويمثل العبور للنظام من حالة إلى أخرى ويمثل بالمخططات بالسهم الواصل بين حالات النظام ويحمل هذا الانتقال نوعين من أنواع المعلومات وهي المؤثر (**أي event**) أو الحدث المؤثر المولد من النظام وقد يكون هذا الحدث داخلي ونتيجة لهذا الحدث فإن النظام سوف يقوم بعمل ما ممثل بـ action وهو الذي به تتغير حالة النظام إلى الحالة التالية.
- تكتب إلى جانب سهم العبور كما هو في الشكل

مخططات حالة الانتقال State Diagrams

بعض الكائنات يمكنها في أي وقت محدد أن تكون في حالة ما. مثلا، كما في نظام الصراف الالي اي أن تكون في إحدى الحالات التالية:

تطبيق عملي:

ATM- STD يراد تحليل وتصميم لنظام ATM وفق STD حيث أن النظام يقدم خدمة الزبائن المصرفية من سحب نقود مالية بالعملة المحلية (2000-5000-10000-20000-30000) أو مبلغ يحدده الزبون بعد إدخال البطاقة وفحص صلاحيتها وفحص الرقم السري المدخل من قبل الزبون كما يمكن تقديم خدمة تحويل مبلغ أو دفع فاتورة أو استعلام عن الرصيد-وفي نهاية أية عمل " transaction" يمكن تقديم وصل بالعملية المنفذة على هذه الـ ATM محدد فيها معلومات الزبون والوقت والتاريخ ومعلومات عن العملية المنفذة .

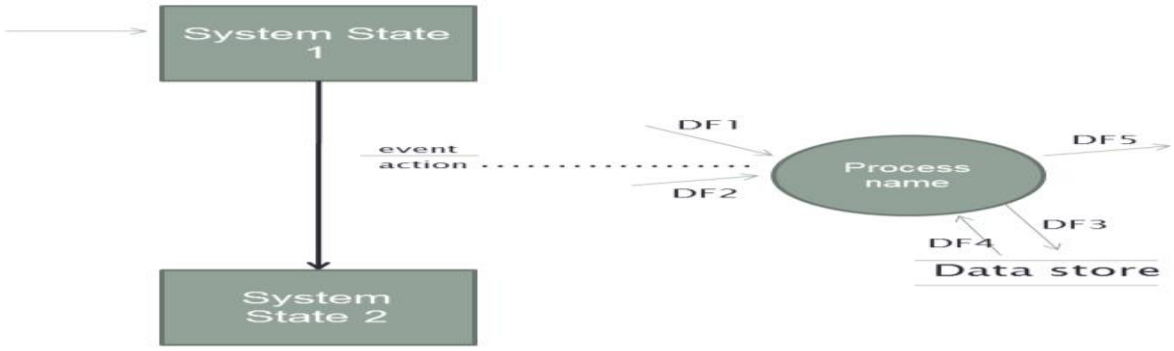


مميزات ومساوئ الـ STD

- يعتبر الـ STD من أحد أهم المخططات التي تصف التتابع التنفيذي لأعمال النظام من خلال البساطة في فهم مخططاته
- يعتبر الـ STD جيداً لتوصفيه تفاعل المستخدم من النظام من خلال استخدامه للأحداث الخارجية
- يعتبر الـ STD غير كافي لعدم وصفه لتتابعات الفرعية لمجريات الأحداث فهو لا يري أي تمييز بين تتابع للأحداث الرئيسية والفرعية

انسياب البيانات مع حالة الانتقال STD with DFD

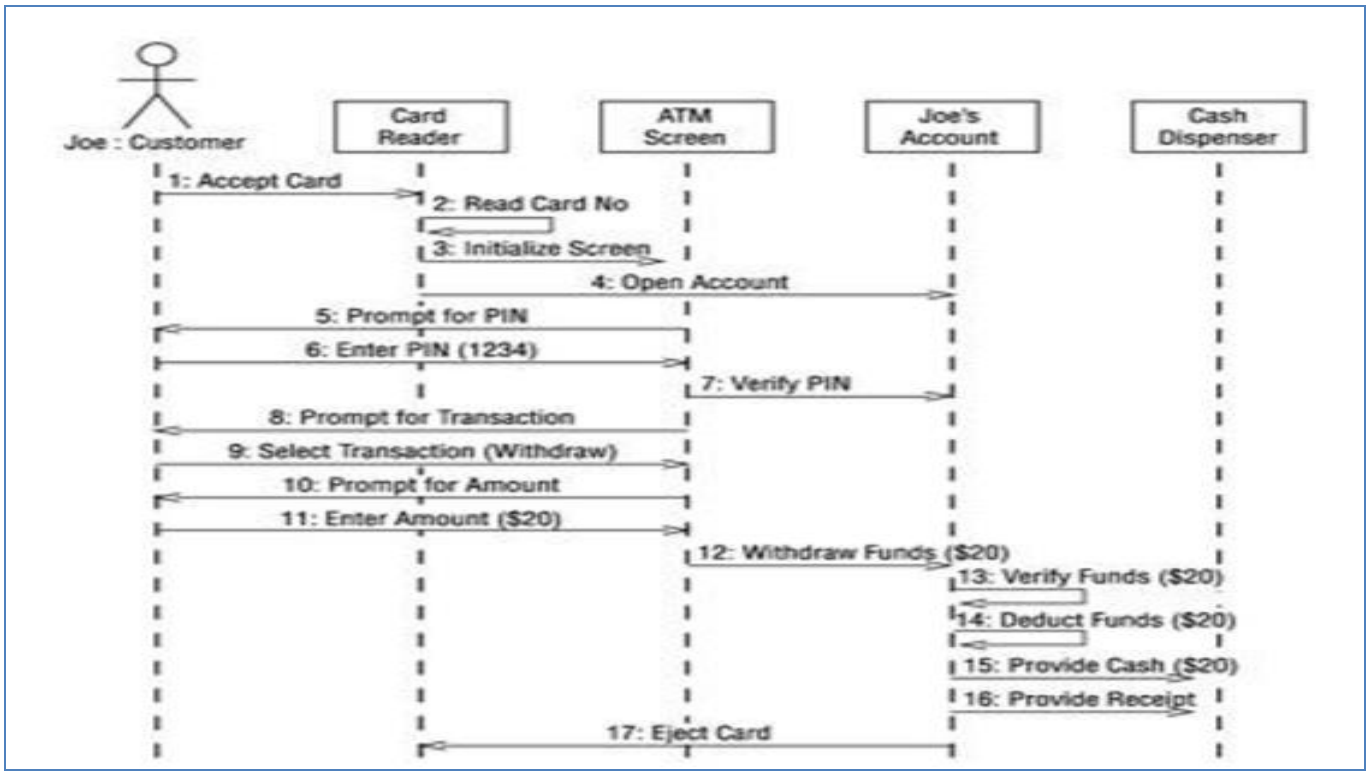
يمكن الاستفادة من خواص لكلا المخططين وايجاد مخطط مشترك يحتوي الـ STD ويستعمل في شرح عملية الانتقال للنظام من حالة إلى حالة أخرى مخطط DDF حيث أن عملية الانتقال قد تستلزم شرح للعمليات التي سوف تؤدي في عملية الانتقال للنظام إلى الحالة التالية ويرسم إلى جانب الانتقال Transition جزء من مخطط DFD ممثل ببعض العمليات التي لها DF و DS وتبين عمليا شرح للاعمال المنجزة في عملية الانتقال .



مخطط التتابع Sequence Diagram في UML

مخطط التتابع في الحقيقة له علاقة مباشرة بمخطط التعاون و يقوم بعرض نفس المعلومات، و لكن بشكل يختلف قليلا. الخطوط المنقطة إلى أسفل المخطط تشير إلى الزمن، لذلك فما نشاهده هنا هو وصف لكيفية تفاعل الكائنات في نظامنا عبر الزمن.

عند بناء وقائع الاستخدام، علينا أن نتعامل مع النظام و كأنه "صندوق أسود"، يقوم باستقبال الطلبات من اللاعب و إرجاع النتائج له. نحن لا يهمنا (حتى الآن) كيف يعمل الصندوق الأسود من أجل تلبية الطلبات.

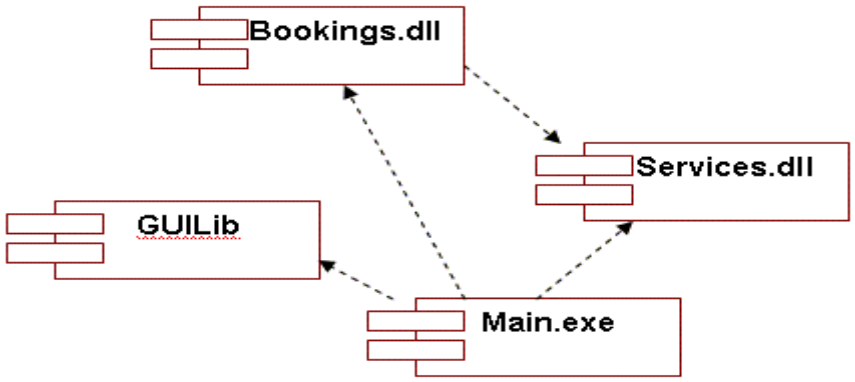


شكل رقم( ): مخطط التتابع لنظام الصراف الالي

لذا و في هذا السياق، نحن ننصح باستخدام مخطط التتابع Sequence Diagram في UML. مخطط التتابع –أو التوالي- مفيد في عدة حالات ، خاصة في مرحلة التصميم. إلا أنه عموما، يمكن استخدام المخطط عند التحليل ليساعدنا في تحليل هذا الصندوق الأسود في النظام. فيما يلي سوف نرى كيف يعمل هذا المخطط:

حال الانتهاء من مخطط التتابع، ستكون مهمتنا سهلة وآلية تماما من أجل وصف التدفق الرئيسي لواقعة الاستخدام. لا حاجة لنا للرسم المضنى لهذه المخططات لكل تدفق بديل أو استثنائي، بالرغم أنها تستحق ذلك في حالة كونها بدائل معقدة جدا أو مثيرة للاهتمام.

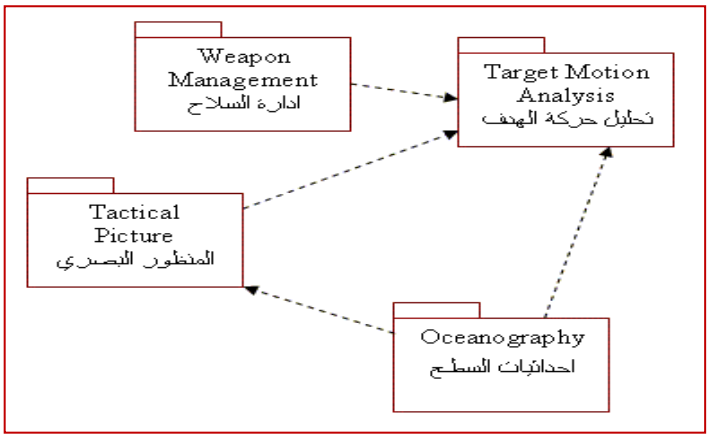
مخططات المكونات Component Diagrams



مخطط المكونات في UML.

يتشابه مخطط المكونات مع مخطط التحزيم - فهو يسمح لنا بترميز كيفية فصل أو تقسيم نظامنا، و كيف يعتمد كل قالب على آخر فيه. عموما، يركّز مخطط المكونات على المكونات الفعلية للبرنامج (الملفات، الراسيات headers، مكتبات الربط، الملفات التنفيذية، الحزم packages) و ليس بالفصل المنطقي أو الفكري كما في مخطط التحزيم. مرة أخرى، سوف نتعمق في دراسة هذا المخطط في فصل معماريات النظام.

مخططات التحزيم Package Diagrams



مخططات التحزيم في UML أي نظام(منظومة) لا يكون صغيرا يحتاج إلى أن يقسّم إلى أجزاء "chunks" أصغر حجما و أسهل للفهم، و تتيح لنا مخططات التحزيم في UML نمذجة هذه الأجزاء بطريقة بسيطة و فعّالة. سوف نتعرّف بكثير من التفصيل على هذا النموذج عند استكشافنا للأنظمة الضخمة من خلال استخدام UML".

النمذجة باستخدام لغة النماذج الموحدة UML

طوّرت هذه التقنية (لغة النمذجة الموحدة) من قبل (Rambaugh &Booch – Rational Software Crop. 1996) من أجل جعل المفاهيم والرموز المستخدمة عالمية ومتداولة بشكل واسع تتناول هذه المنهجية في التحليل مواضيع مختلفة فهي تحلل النظام من خلال الـ Objects– المتواجدة أو المولفة له (النظام عبارة عن تفاعل مجموعة من الـ Objects) وبالتالي هنالك مجموعة من المخططات التي تعكس عمل النظام وتتناسب والمفاهيم الجديدة OOP's.

الارشادات للعمل في UML

ويتم العمل معUML من خلال النصائح التالية:

1- تطوير النظام البرمجي بشكل متكرر Develop software iteratively

2- ادارة المتطلبات النظام بشكل دقيق وصحيح Manage requirements

3- استخدام العناصر الجاهزة في بناء معمارية Use component-based architectures

4- بناء نموذج مرئي Visually model software

5- التأكد من البرامج مرحلة بمرحلة Verify software quality

6- امكانية اجراء التعديلات على النظام وامكانية الصيانة له Control changes to software

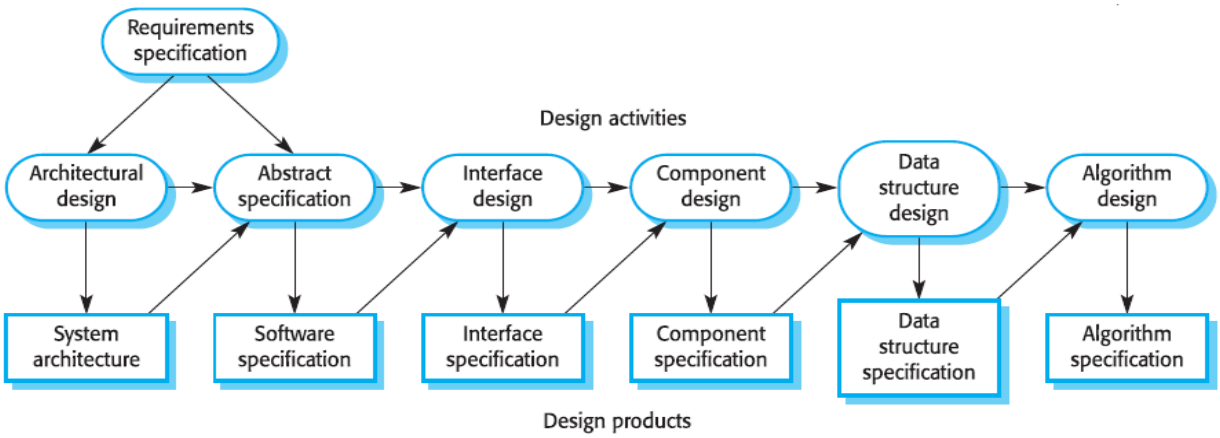
الفصل الخامس

مرحلة تصميم البرمجيات

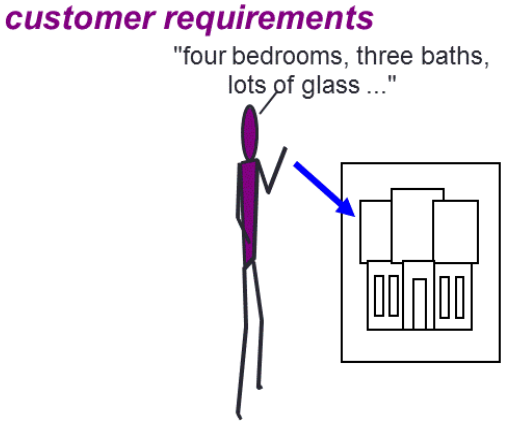
(Software Design Phase)

مرحلة التصميم لبرنامج هي عملية تطوير نماذج متطلبات ونماذج تحليل النظام الى نماذج تصميم النظام والتي فعلا تمكننا من بناء هيكل البرنامج و أجزاءه و كيفية ترابطها مع بعضها البعض، ينتج عن ذلك مجموعة من الملفات و النماذج و الرسومات البيانية التي يمكن منها برمجة وكتابة شيفرة البرنامج "بالكامل".

معالجة تصميم البرمجيات The Software Design Process



و تنقسم مرحلة التصميم إلى مراحل عديدة من أهمها:



1- تصميم البنية أو المعمارية Architectural Design

2- توصيفات مجردة Abstract specification

3- تصميم واجهة الاستخدام Interface design

4- تصميم مكونات النظام(الاجزاء) Component design

5- تصميم هيكلة البيانات Data structure design

اولا: تصميم البنية أو المعمارية (Architectural Design).

معمارية البرمجية هي عملية ترتيب لأجزاء البرنامج بطريقة معينة ومرتيب و تنظيم ترابط هذه الأجزاء مع بعضها البعض. غالبا لا تتطلب هذه المرحلة الابتكار، فهناك أنواع معروفة و محددة من الهياكل. كل ما يُستلزم في هذه المرحلة هو تصميم أو اختيار الهيكل المناسب للبرنامج. يعتمد اختيار الهيكل على نوع البرمجية و على دراسة جوانب أخرى مهمة اهمها:

1- الأداء (performance)

2-الأمان والحماية (security and safety)

4-سهولة الصيانة (Maintainability).

1- تصميم معمارية التخزين أو المخزن (Repository Architecture)

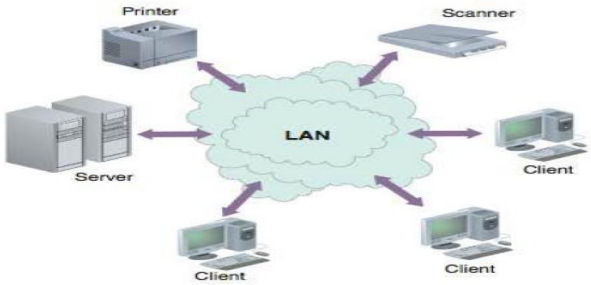
و يستخدم في البرامج التي تعتمد على قاعدة (أو قواعد) بيانات كبيرة (Database).

سوف نختصر دراستنا على نماذج تحليل وتصميم قواعد البيانات والتي سبق وان شرحناها بالفصل السابق نمذجة تحليل النظام. تم شرحها مسبقا.

2- **) معمارية الطبقات (Layered Architecture)**

ويستخدم في البرامج التي تنقسم إلى أجزاء واضحة (طبقات). و كل طبقة تعتمد على ما تحتها من طبقات.

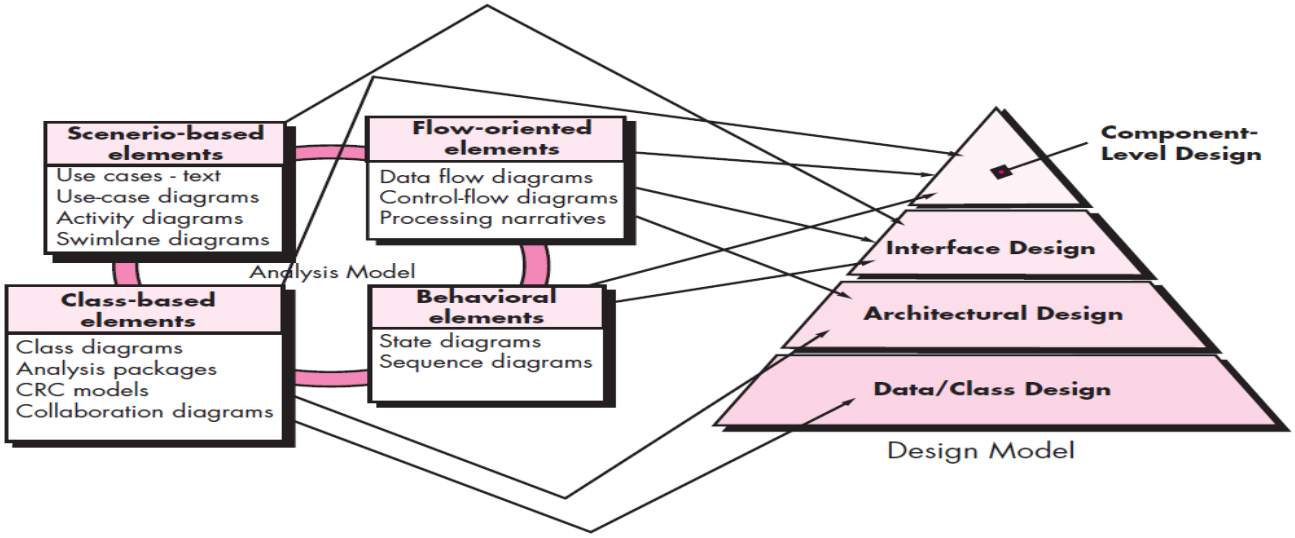
مثال: برامج تحويل و إرسال البيانات عبر الشبكة. فتحويل البيانات يستلزم مراحل و كل مرحلة تعتبر طبقة.



3- **ج) معمارية الخادم-المخدوم (أو الزبون) (Client-Server Architecture)**

و يستخدم في البرامج التي تعمل على أكثر من جهاز أو موزعة على شبكة. مثال: الانظمة الموزعة وهي التي تنقل من مكان الى اخر ومن مدينة الى اخرى.

ثانيا: توصيفات مجردة (مكونات الاصناف): نمذجة الكائنات الموجهة



ثانيا: تصميم قواعد البيانات

ثالثا: تصميم واجهة الاستخدام (GUI Design) :

واجهة الاستخدام هي واجهة البرنامج التي يتعامل معها المستخدم. وتشمل بالأساس الجزيء المرئي الذي يتكون عادة من نوافذ و أزرار و حقول كتابة و لون الخلفية و لون الخط... الخ.

يتعلق هذا القسم بالجزء الفني و جمالي من البرنامج. كم يهتم بسرعة استجابة الواجهة و إمكانيتها إعطاء معلومات واضحة و كافية للمستخدم. وهناك دراسات كثيرة في هذا المجال تشمل دراسة الجانب النفسي للإنسان و تأثره بالألوان و بطريقة ترتيب الأشياء.

على سبيل المثال، من النصائح المعروفة في هذا المجال هي:
لا تستخدم جميع ألوان قوس قزح في واجهة برنامجك! .
من المفضل استخدام لونين إلى أربع ألوان مختلفة كحد أقصى.

3- تصميم الشاشاتSecreens Design

تصميم شاشات النظام : يقصد بشاشات النظام مجموعة المعلومات التي تظهر في نظام المعلومات، واللازمة لتوضيح وظائف النظام ويتم تصميم هذه الشاشات وفقا لخوارزميات تدفق المعلومات .
يحتوي كل نظام على عدة شاشات ، وهذه الشاشات يمكن أن يكون لها استخدام مزدوج، كوحدة مدخلات ومخرجات في نفس الوقت ، كما موضح في الشكل ادناه .

- **تصميم واجهة الدخول الى النظام**

شكل رقم( ) تصميم واجهة الدخول لنظام منشاة المياه والكهرباء

تصميم الواجهة الأولى للنظام
وهي الواجهة التي تظهر مباشرتا بعد واجهة الدخول ، وتحتوي على خيارات البرامج ووظائف النظام وتتفرع من الواجهة الرئيسية عدة واجهات فرعية مثل:

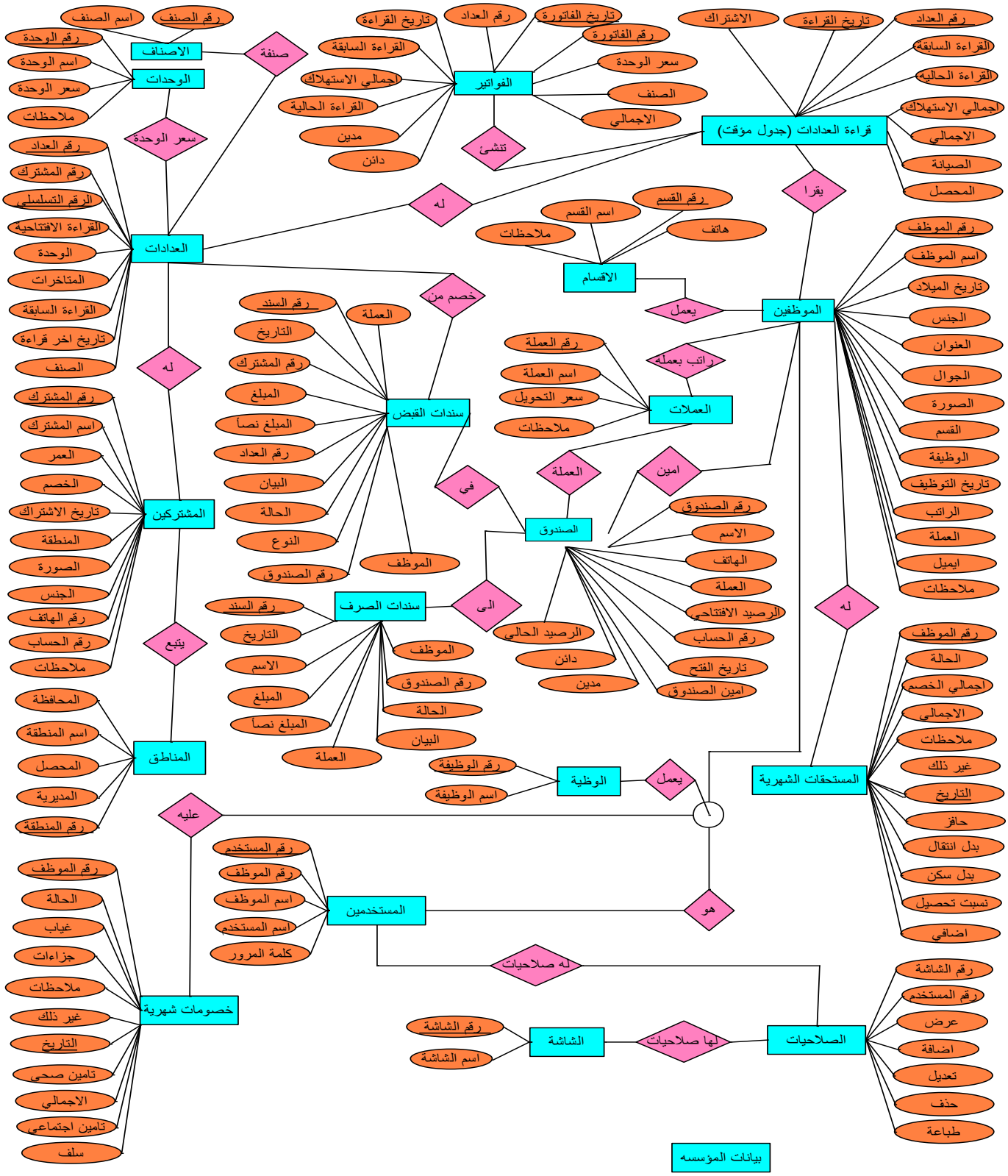
- **واجهة إدخال البيانات**
- **واجهة البحث عن بيانات او معلومات**
- **واجهة تعديل البيانات**
- **واجهة اعداد واختيار التقارير- واجهة الطباعة واعدادها**
- **واجهة المساعدة والتعليمات لاستخدام النظام**
- **واجهة صيانة النظام واجراء النسخ الاحتياطية**
- **واخيرا واجهة الخروج من النظام بامان.**

تصميم الواجهات او النوافذ المتفرعة من الرئيسية الطبقة الثانية
تحوي على واجهات فرعية متفرعة
- يصدر منها رسائل الخطاء من سوء الاستخدام او الرسائل التحذيرية او التنبيهية او رسائل التغذية المرتدة والاطمئنان على ان تنفيذ المهام المطلوبة قد تم او ان العملية فشلت.
- واجهة الحوار لتحديد مسار التنقل بين النوافذ سواء لحفظ الملفات او فتح ملفات او تحميل ملفات من مجلدات او مخازن داخل اسطوانات النظام او من خارجه.

تصميم واجهات او نوافذ او صناديق الحوار الطبقة الثالثة
خاصة بتنبيه مستعملي النظام لمجموعة الأخطاء التي قد يرتكبونها أثناء التشغيل، او المطالبة بمزيد من الاعدادات او توفير برمجيات ومستلزمات لتشغيل او اجراء مهمة خارجة عن مكونات النظام .

تصميم واجهات او نوافذ او صناديق الحوار الطبقة الرابعة
تحتوي على نص يدل على إرشادات مستعملي النظام بمتابعة العمل أو الخروج من النظام.

5-4 التصميم المنطقي للكينونات والعلاقات ER-Diagram



الشكل(1-3) يوضح العلاقات بين الكينونات للنظام ER-Diagram

ثانيا: التصميم الفيزيائي للنظام (التصميم القابل للتنفيذ بواسطة الحاسب الالي)

6-4 التصميم الفيزيائي لجداول قاعدة البيانات

من عملية التصميم الفيزيائي لكينونات قاعدة البيانات والعلاقات بينها استنتج الفريق خصائص ومواصفات مخطط وجداول البيانات الفيزيائية وتحديد حقولها وسجلاتها والعلاقات بين كل الجداول وانواع بياناتها، بحيث تنساب البيانات وتتنقل من جدول الى اخر ، ويتم تحقيق التفاعل مع هذه القواعد من اضافة وتعديل وحذف واستعلام عن البيانات المحددة للنظام. وقد استخدم الفريق التصميم الفيزيائي لقاعدة بيانات النظام كمايلي:

اولا:المخطط الفيزيائي لقاعدة البيانات

الشكل(2-3) يوضح العلاقات بين الكيانات ER-Diagram

ثانيا: تصميم الجداول الفيزيائي

1- جدول المناطق

DESKTOP-8JBEG7H....G2 - dbo.REGIONS

Column Name	Data Type	Allow Nulls
ID	int	☐
NAME	nvarchar(50)	☐
GOVERNORATE	nvarchar(50)	☐
THE_DIRECTORATE	nvarchar(50)	☐
ID_COLLECTOR	int	☐
		☐

الجدول (3.1) يوضح جدول المناطق

2.جدول المشتركين

DESKTOP-8O78RVD...- dbo.CUSTOMER | DESKTOP-8O78RVD...G2 - dbo.REGIONS

Column Name	Data Type	Allow Nulls
ID	int	☐
NAME	nvarchar(100)	☐
AGE	int	☐
SUBSCRIPTION_DATE	date	☐
ACCOUNT_NUMBER	int	☐
DISCOUNT	float	☑
NOTES	text	☑
ID_REGIONS	int	☐
IMAGE_CUSTOMER	image	☑
		☐

الجدول (3.2) يوضح جدول المشتركين

3. جدول الفواتير /والقراءات

DESKTOP-8O78RVD...- dbo.CUSTOMER DESKTOP-8O78RVD...G2 - dbo.REGIONS

Column Name	Data Type	Allow Nulls
ID	int	☐
DATE_BILL	date	☐
READING_DATE	datetime	☑
ID_COUNTERS	int	☐
ID_REG	int	☑
NAME_CUSTOMER	nvarchar(100)	☐
PREVIOUS_READING	float	☐
CURRENT_READING	float	☐
DIFFERENCE_READER	float	☐
PRICE_OF_THE_UNIT	float	☐
ARREARS	float	☐
SUBSCRIPTION	float	☐
MAINTENANCE	float	☐
TOTAL	float	☐
NOTS	text	☑
DATE_PAY	date	☑
TOTAL_PAY	float	☑
TOTAL_PAY_A	float	☑
TOTAL_PAY_L	float	☑
ok	bit	☑
PUSH	bit	☑
		☐

الجدول (3.3) يوضح جدول الفواتير والقراءات

4. جدول العدادات

DESKTOP-8O78RVD....LLING2 - dbo.BILL DESKTOP-8O78RVD...- dbo.COUNTERS

Column Name	Data Type	Allow Nulls
SERIAL_NUMBER	int	☐
ID	int	☐
ID_CUSTOMER	int	☐
READING_READERS	float	☐
PRICE_OF_THE_UNIT	float	☐
ID_ITEMS	int	☐
NOTES	text	☑
PREVIOUS_READING	float	☐
UNIT_ID	int	☐
NAME_CUSTOMER	nvarchar(200)	☐
ARREARS	float	☐
DATE_END_READ	datetime	☑
		☐

الجدول (3.4) يوضح جدول العدادات

5. جدول الصناديق

DESKTOP-8O78RVD....LLING2 - dbo.BOX

Column Name	Data Type	Allow Nulls
BOX_ID	int	☐
BOX_NAME	nvarchar(100)	☑
BOX_PHON	nvarchar(50)	☑
BOX_ACCOUNT_NUMBER	int	☑
BOX_DATE_OPEN_BOX	datetime	☑
BOX_OPEN_BALANCE	int	☑
BOX_THE_CURRENCY	nvarchar(50)	☑
BOX_MAXIMUM	int	☑
BOX_DEBIT	float	☑
BOX_CREDITOR	float	☑
BOX_Balance	float	☑
BOX_EMP_ID	int	☑
BOX_NOTE	text	☑
		☐

الجدول رقم (3.5) يوضح جدول الصناديق

رابعا: تصميم المكونات الاجزاء

خامسا: تصميم هياكل البيانات

سادسا: تصميم الخوارزميات.

الوحدة الخامسة

The System Implementation Phase مرحلة تنفيذ النظام

1-6 كتابة الكود البرمجي System Coding

2-6 اختبار النظام System testing

3-6 تركيب النظام System Installment

4-6 تقويم النظام System Evaluation

5-6 توثيق النظام System Documentation

1-6 كتابة الكود البرمجي System Coding

المقدمة:

تبدا هذه المرحلة فور الانتهاء من مرحلتي التحليل والتصميم وقد تكوف جزء من مرحلة التصميم خاصة عند استخدام بعض النماذج (النموذج التزايدي ، نموذج التمثيلي الاولي) ، وخلال هذه المرحلة يتم فيها ترجمة كل نماذج التصميم الى اسطر برمجية تكتب بأحد لغات البرمجة عالية المستوى استخدام لغات البرمجة او ادارة قواعد البيانات المناسبة لعملية كتابة الاسطر البرمجية وتنفيذ اهداف ومتطلبات النظام المحللة والمصممة بدقة عالية وموضحة تماما لكل الفرق البرمجية ، حتى يضمن جودة البرامج وسهولتها في الفحص والاختبار والصيانة من اول مرحلة الترميز دون لرجوع الى كتابتها مرات متعاقبة او استهلا الجهد والزمن في عملية الاضافة او الحذف لبعض مطاليب غامضة قد يترتب عليها فشل النظام وعدم الاستفادة منه وتبديد كل الجهود التي بذلت في المراحل الثلاث من مراحل التطوير.

كتابة الكود البرمجي للنظام (الترميز) Coding

تكتب الإجراءات في كتيب يسمي **دليل الإجراءات :**

يجب أن يصمم هذا الدليل بطريقة مرنه لكي يسهل تعديله عند اللزوم ويحتوي على معلومات تفصيلية خطوة بخطوة بخصوص كيفية إجراء كل عملية أو نشاط معين .

2.7 مزايا الإجراءات المكتوبة

1. تعزيز الاهتمام بالنظام.
2. توحيد أسس العمل طبقا للمعايير القياسية.
3. تسهيل الإشراف والرقابة على العمل.
4. تعتبر أساس لتدريب العاملين على خطوات تنفيذ النظام.
5. تحديد دور الأفراد ومسئولياتهم.
6. تسهيل تطوير الإجراءات.
7. استمرار العمل في حالة تغيب العاملين.

1.تعريف الأدوات المساعدة في البرمجة CASE Tools

CASE tools= Computer aided software engineering

أو أدوات هندسة البرمجيات باستخدام الحاسوب ، وهي مجموعة برمجيات تأخذ منك تصميمك النظري للبرمجيات وتحوله إلى شكل flow charts and design diagrams ، ومن ثم تساهم بصورة شبه آلية في إنتاج الكود الموافق للتصميم، وبالتالي تساهم في تقليل زمن دورة تطوير البرمجيات software development cycle وتزيد من قابلية إعادة استخدام البرمجيات software reusability. من أشهر هذه البرمجيات Rational Rose و Microsoft Visio، كما أن العديد من بيئات التطوير أصبحت تتنافس في تقديم ميزات هندسة البرمجيات باستخدام الحاسوب ،منها مثلا Visual studio.Net و Borland JBuilder X.

ويمكن تعريفها كذلك على أنها برمجيات البرمجة Programming Softwareوهي البرمجيات المساعدة في عملية إنتاج البرمجيات Software Development نفسها؛ و من أمثلتها برمجيات إدارة قواعد البيانات Database Management Systems DBMS وبرمجيات بيئة البرمجة المتكاملة Integrated Development Environment IDE، و التي تحتوي بدورها على مولد المترجمات Compiler و محرر اللغة البرمجية Editor و الأدوات البرمجية Tools المختلفة التي يحتاجها المبرمج أثناء عمله.

ومع تطور علوم الحاسوب فقد ظهرت أساليب حديثة لتطوير وبناء نظم المعلومات. والهدف من هذه الأساليب الحديثة هو اختصار الوقت الذي تتطلبه عملية بناء نظم المعلومات، وتيسير عمليات كتابة البرامج ومراجعتها والتأكد من تأديتها للوظائف المتوقعة منها.

ومن أبرز اتجاهات هذا التطور هو ظهور أدوات وبرمجيات هندسة البرمجيات بمساعدة الحاسوب أو ما يعرف اختصارا بأدوات "كيس" (CASE Tools). والهدف الأساس من أدوات كيس هو أتمتة عمليات توليد البرمجيات وتطوير التطبيقات. ويشمل ذلك أتمته جميع المراحل والتي تشمل(تحديد الاحتياجات، التحليل، التصميم، كتابة البرامج ، اختبار البرامج وصيانتها) .

إذن فإن استخدام أدوات كيس لا يعني الاستغناء عن أي مرحلة من مراحل تطوير نظم المعلومات، بل هو وسيلة لاستخدام الحاسوب في تطوير تطبيقات الحاسوب بهدف اختصار زمن التطوير وتيسير عمليات توليد وثائق النظام وتنظيم عمليات الفحص والصيانة.

2. مميزات وسلبيات استخدام أدوات كيس

1.2 مزايا استخدام أدوات كيس

ويمكن تلخيص أهم مزايا استخدام أدوات كيس في التالي :

أ) سرعة إنتاج البرامج

تتضمن أدوات كيس في العادة مكتبةً كبيرة من البرامج الجاهزة التي تؤدي وظائف مختلفة. ومن واقع نتاج تحليل النظام وتحديد متطلبات النظام والعمليات المطلوبة فيه يمكن لأدوات كيس أن تنتج نسخة أولية من البرامج المطلوبة بسرعة. هذه البرامج يمكن أن تكون الأساس الذي تبنى حوله النظم التطبيقية المطلوبة.

ب) خفض تكاليف التطوير

لاختصار زمن التطوير وسرعة إنتاج البرامج المطلوبة، يمكن أن نتوقع خفض التكلفة بما يشمله من تكلفة كتابة البرمجيات وتعديلها وفحصها وكذلك تكلفة إنتاج الوثائق وغيرها.

ت) الالتزام بمعايير ثابتة للتطوير

إن استخدام أدوات كيس يضمن الالتزام بمعايير واضحة وثابتة لجميع مراحل التطوير. وذلك لأن هذه الأدوات تتطلب توفر معلومات محددة منظمة ومرتبة وفق هياكل خاصة، ولا يمكن لهذه الأدوات أن تنتج البرامج والتطبيقات المطلوبة إلا بعد توفر المدخلات المطلوبة.

ث) تيسير تطوير الأنظمة الكبيرة

ونظرا لأن أدوات كيس يمكنها توليد البرامج بطريقة هيكلية منظمة، فإنه يسهل بواسطتها التعامل مع الأنظمة الكبيرة المعقدة. وهنا يمكن بواسطة هذه الأدوات تقسيم النظام الكبير إلى وحدات مترابطة يمكن توزيعها على أكثر من مبرمج، وعلى أن يجري تكاملها وربطها في مراحل لاحقة.

ج) تكامل التوثيق

بينت التجارب أن توثيق النظام يكون في العادة أضعف جوانب تطوير الأنظمة. وعندما يتم تعديل البرامج وتحديثها، فإنه يغفل غالبا تحديث الوثائق حتى تعكس التغيرات التي حدثت على البرامج . ولكن باستخدام أدوات كيس فإن الوثائق الفنية يمكن توليدها آليا وهي جزء أساسي من المخرجات. وعند تعديل البرامج يمكن دائما توليد وثائق فنية حديثة.

2.2 سلبيات استخدام أدوات كيس

وفي مقابل المزايا التي توفرها أدوات كيس ، فإن هناك عدداً من السلبيات التي يجب عدم إغفالها عند اتخاذ القرار المتعلق باستخدام هذه الأدوات من عدمه. ويمكن تلخيص **أبرز سلبيات أدوات كيس في التالي:**

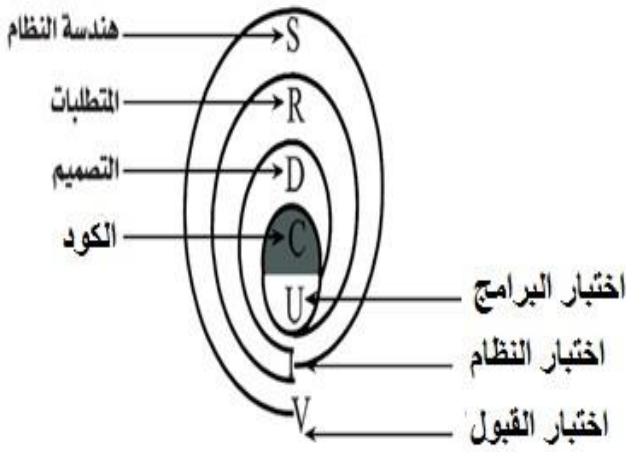
- صعوبة تعلم استخدام هذه الأدوات واحتياجها لفترات تدريب طويلة.
- قلة أعداد المختصين المتمرسين في تطوير التطبيقات باستخدام أدوات كيس.
- صعوبة تطوير بعض التطبيقات الخاصة غير التقليدية بواسطة أدوات كيس.
- ارتفاع التكلفة المبدئية لاستخدام أدوات كيس نسبيا بسبب حداثة هذه البرمجيات وقلة عدد الشركات التي تنتجها. ولكن عند استخدام هذه الأدوات في تطوير العديد من الأنظمة فإن التكلفة إجمالا ستنخفض.

2.5 مرحلة اختبار النظام

1.2.5 مستويات اختبار النظام

يتم ذلك وفقا لمستويات خمسة وهي :

أ- إختبارالبرامج:



المستوى الأول : اختبار أجزاء البرامج .

المستوى الثاني : اختبار البرامج كلها .

ب- اختبار النظام :

المستوى الثالث : اختبار أجزاء النظام .

المستوى الرابع: اختبار تكامل النظام .

ج- اختبار القبول :

المستوى الخامس : الاختبار النهائي للبرامج

2.2.5 إجراءات الاختبارات

الإجراءات الواجب اتخاذها للاختبارات هي:

1- وضع خطة الاختبارات.

2- وضع شروط الاختبارات الفنية للنظام.

3- إعداد بيانات الاختبارات والنتائج المتوقعة.

4- وضع الجداول الزمنية للاختبارات.

5- تنفيذ الاختبارات ومراقبة النتائج وأداء النظام.

6- تقويم مشاكل اختبارات النظام.

3-5 تركيب النظام System Installment

1-3-5 مرحلة تحويل النظام (تشغيل النظام)

في هذه المرحلة يتم نقل وتشغيل النظام في المؤسسة وبالتالي التحويل للعمل بالنظام الجديد.

في كثير من الحالات يمكن أن تنقل ملفات البيانات من النظام القديم إلكترونيا إلى النظام الجديد باستخدام بعض أنواع نظم البرمجيات في توجيه التحويل، وفي بعض الاحيان تكون عملية النقل مستحيلة.

- يتم التحويل إلى النظام الجديد باستخدام إحدى طرق التحويل المعروفة حسب طبيعة النظام .

أ) التخطيط للتحويل

يخطط المحلل لعملية التحويل ويشرف عليها، ويقوم بمراعاة مايلي:

1- وضع الجدول الزمني للتحويل.

2- حصر بعض المؤشرات عن حجم عمل التحويل وما تحتاجه من تقنية وقوى بشريه.

ب) أنواع التحويل المختلفة

1- التحويل الفوري/ المباشر.

2- التحويل بالتوازي.

3- التحويل التدريجي.

التحويل الفوري : Direct Conversion

- هو توقيف العمل بالنظام القديم وبدء العمل بالنظام الجديد وذلك باعتبار أن النظام الجديد تم اختباره ولم يفشل، وهذه الطريقة مناسبة لتحويل النظم البسيطة غير المعقدة ، ولكن تتطلب بعض الظروف استخدام هذه الطريقة للتحويل مثل :

1- عجز النظام القديم عن تأدية عملة.

2- كون قاعدة البيانات الجديدة مختلفة تماما عن قاعدة البيانات القديمة.

3- كون واحدة أو أكثر من أجهزة النظام الجديد لا تعمل مع النظام القديم.

التحويل بالتوازي : Parallel Conversion

- وفيه يستمر العمل بالنظام القديم حتى يتم التأكد من أن النظام الجديد يعمل بطريقة مرضية مع استخدام النظام الجديد لإجراء العمليات المطلوبة منه.
- كما أن البيانات تعالج في كلا النظامين في نفس الوقت فيعمل النظامان معا لمدة دورة عاديه للنظام .

التحويل التدريجي : Stepped Conversion

- ينفذ النظام على أساس تدريجي، ووفقا لذلك لا يتم إحلال النظام الجديد بالكامل محل النظام القديم ولكن يتم إحلال جزء منه، ويظل الباقي يعمل وفقا للأسلوب القديم حتى يثبت نجاح التنفيذ في الجزء الذي تم إحلاله .
- كذلك يتم تطبيق النظام الجديد على عدد محدود من المستفيدين ثم يتم تعميمه بعد التأكد من نجاحه.

مثال :

يتم تطبيق نظام المرتبات والأجور جزئيا على قسم واحد وبعد أن يثبت نجاحه يتم تعميمه على بقيه الأقسام .

4-5 مرحلة تقويم النظام System Evaluation

يجب أن يقوم النظام الجديد تقويماً شاملاً وذلك بعد تشغيله لفترة قصيرة، ويكون التقويم من ناحيتين:

التقويم الفني Technical Evaluation

أي فحص أداء النظام ومخرجاته وذلك بغرض التحقق والتأكد من أن النظام :

1- حقق الأهداف الموضوعة له.

2- وأنه ليس هنالك انحراف عن النتائج المتوقعة.

3- تحديد إمكانية احتياجه إلى تعديلات أو تغييرات.

التقويم المالي Financial Evaluation

وذلك بتحليل التكلفة مقابل الفوائد من النظام ومدى تحقيق النظام للأهداف الاقتصادية الموضوعة له وعملية التقويم هذه الوجه الآخر لدراسة الجدوى .

5-5 مرحلة توثيق النظام System Documentation

التوثيق هو وصف كتابي للنظام وأهدافه وأجزائه وإجراءات تشغيله مدعوما بالوثائق والمستندات والرسومات الإيضاحية والجداول الو صفيه .

وتبدأ عملية التوثيق مع بداية المشروع ولا تنتهي بنهايته ، بل تظل ملازمة للنظام طوال فترة عمله وتشغيله

أهمية التوثيق

إذا تم توثيق النظام توثيقا جيدا ودقيقا فإنه :

1- يعد مرجعا عاما لإدارة المنشأة.

2- يوفر لمحلل النظم مصدرا قيما للمعلومات لتطوير النظام وصيانته باستخدام وثائقه.

1- يساعد مستخدميه على فهم النظام وتتبع إجراءاته.

محتويات توثيق النظام

(1) **المقدمة :**

- أهداف النظام الجديد وفوائده وخواصه.
- طرق جمع المعلومات وتحضيرها للمعالجة .
- وصف عمليات النظام .
- توضيح إمكانية تحويل النظام الجديد وتطويره مستقبليا .

(2) مخطط النظام : System Flowchart

- توضيح سير المعلومات داخل النظام وخارجه، مع بيان العلاقات المنطقية بين البرامج المختلفة .

(3) وصف الملفات : Files Specifications

- يتم ذلك في نماذج وصف الملفات .
- تحتوي هذه النماذج على :

اسم الملف / فترة استخدامه / طريقة تنظيمه / وسط التخزين / حجم الملف / المفاتيح المستخدمة / الإجراءات الواجب اتباعها عند حدوث خلل في الملف .

- معلومات عن السجلات / اسم السجل / وصف السجل / طول السجل .

(4) وصف السجلات : Records Specifications

تستخدم نماذج وصف السجلات والتي تحتوي على :

- اسم الملف المستخدم للسجل .
- الحقول الموجودة في السجل، اسم الحقل / طوله/نوعه/موقع الحقل.

- يعبأ نموذج واحد أو أكثر لكل سجل .

- ترقم النماذج بشكل تسلسلي .

(5) وصف التقارير : Reports Specifications

تستخدم نماذج وصف التقارير لتحديد :

- الهدف.
- الحجم : عدد النسخ.
- الورق : نوعه.
- التحكم : عدد الأسطر في الصفحة .
- السريان : إن وجد.
- التوزيع : كيفية التوزيع .

(6) متطلبات البرامج من المعدات:

تستخدم نماذج خاصة تحوي :

- اسم البرامج : توضيح أسماء البرامج الداخلية في النظام حسب ترتيبها في مخطط النظام.

- حجم الذاكرة اللازمة.

- الوقت اللازم لتنفيذ البرامج .

(7) توثيق البرامج : Programs Documentation

عند توثيق البرامج يجب مراعاة الأمور التالية :

1- وصف كافة العمليات والمدخلات والمخرجات لكل برنامج على حده.

2- عند إجراء أية تعديلات على البرامج يجب إحداث هذه التعديلات على نماذج توثيق البرامج.

3- كل برنامج يوثق له بالطريقة التالية :

أ- مخطط البرنامج : Program Flowchart

تستخدم نماذج خاصة يوضح فيها :

- اسم البرنامج.

- رقم البرنامج .

- اسم المبرمج .

- تاريخ كتابة البرنامج .

- رقم الصفحة .

- تحديد العمليات التي يقوم بها البرنامج .

ب- وثيقة التنفيذ : Run Chart

يحدد فيها:

- اسم البرنامج .
- المدخلات / المخرجات .
- هدف تنفيذ البرنامج .
- علاقة هذا البرنامج مع البرامج الأخرى .
- شروط تنفيذ هذا البرنامج .
- اللغة المستخدمة لكتابة هذه البرنامج .

ج- وثيقة وظائف البرنامج : Program Functions

تحتوي هذه الوثيقة على :

- الإجراءات الأولية / الخطوات الأولية لتنفيذ البرامج .
- مدخلات الشاشة .
- الجداول المستخدمة .
- إجراءات المعالجة .
- رسائل الشاشات .

د- وثيقة التحكم بالبرنامج : Program Control

- توضح كافة رسائل الأخطاء الصادرة من البرنامج .
- التأكد من مراجعة السجلات (سجلات الملف الرئيسي) عند تحديثها .
- توضيح كافة الرموز المستخدمة في البرنامج .

ملاحظة: تدريب المستخدمين على النظام الجديد تعتبر خطوة هامة في تشغيل البرنامج تحدث عن ذلك من وجهة نظرك.

الخلاصة

ناقشنا في هذه الوحدة مرحلة تطبيق النظام، وبيّنا أنها عملية يتم من خلالها تهيئة الموقع في الجهة المستخدمة لتركيب الأجهزة والبرمجيات الخاصة بالنظام الجديد، واختبار مستويات النظام الخمسة، وهي تشمل البرامج، والنظام، والقبول، إضافة إلى التحول من النظام القديم إلى النظام الجديد بهدف زيادة فعالية النظام وقدرته.

ثم خطونا إلى مرحلة تقويم النظام بهدف التأكد من أن النظام يحقق الأهداف الموضوعة له ويكون التقويم فنياً ومالياً ، لكي نقدم النظام إلى الجهة المصمم لها في صورة موثقة توثيقا جيدا يضمن عدم فقدان المعلومات التي تخصه.

تدريب(1)

مايجب أخذه بعين الاعتبار عند وضع مواصفات أجهزة شبكات الاتصال الحاسوبية هو:

1. أن تكون وفقاً للمواصفات العالمية.
2. سرعة نقل البيانات
3. عدد الدوائر الكهربائية ونوعها.

4. مواصفات المودم وسرعته.
5. المسافة بين نقاط الاتصال على الشبكة.
6. تكامل النظام.
7. تكاليف تركيب الشبكة وأجهزتها وصيانتها.

تدريب(2)

عند تصميم نظام جديد وتطويره يكون النظام واضحا ومفهوما بشكل جيد للذين صمموه وطوروه أكثر من غيرهم، وبما أن النظام صمم لجهة معينة مستخدمة له، وسوف يتم تشغيله من قبل أشخاص آخرين وربما يعمل عليه مبرمجون جدد، فأنه توجب عليك كل هذه العوامل التفكير في التدريب و كيفية وضع الخطط له، وتنفيذها وفق جداول زمنية محددة.

المصطلحات

- **اختبار النظام System Testing**
اختبار صحة البرامج والملفات وقواعد البيانات.

- **التحول Conversion** الانتقال من تطبيق النظام القديم إلى نظام آخر جديد.

الاسئلة:

1) ما هي مكونات مرحلة تحويل النظام ؟
2) ما هو التخطيط للتحويل؟
3) ما هو التحويل الفوري ومتي يستخدم ؟
4) ما هو التحويل بالتوازي ؟
5) ما هو التحويل التدريجي ؟
6) ما هي عناصر التقويم للنظام؟
7) ما هي أهمية توثيق النظام ؟
8) اذكر محتويات توثيق النظام.
9) ما هي مراحل تطبيق النظام؟
10) اذكر مستويات مرحلة اختبار النظام.

الوحدة السادسة

مرحلة صيانة النظام والدعم

The System Maintenance and Supporting

في هذه المرحلة يتم إدخال التعديلات على النظام بعد أن يصبح نظاما شغال في المؤسسة المالكة للنظام.

- عادة يتم إنفاق من **[50% - 70%]** من جهد البرمجة الكلي علي الصيانة .

يحتاج النظام للصيانة لسببين هما :

1- لإصلاح العيوب في النظام عند تسليمه.

2- لإجراء التعديل والتغير لوظائف النظام نظرا للطبيعة المتغيرة لبيئة الأعمال.

المخرجات بعد عملية الصيانة : هو نظام مجدد وتوثيق مستخدمين مجدد وبرامج أجريت لها مراجعات

الأدوات الأساسية : قاموس البيانات ، ورسومات تدفق البيانات ، ومواصفات العمليات ، ونماذج البيانات ، ونماذج النظام ،وخرائط تدفق النظام وصيغ تصميم المخرجات والمدخلات .

الأفراد والمهام :

1- يرسل المستخدمون للنظام الى المحلل بوجود مشكلة ما أو حاجة تغيير مقترح علي النظام.

2- يعد المحلل نموذجا لتقويم تأثير التعديل.
3- يتم اتخاذ القرار بما إذا كان التعديل سينفذ أم لا.
4- إذا حدثت موافقة بالتغيير، يقوم المحلل بتعديل توثيق النظام كله لعكس هذا التغيير.
5- يقوم المبرمجون بتعديل البرامج.
6- يختبر فريق الاختبار البرامج المعدلة .
7-يقوم فريق التكامل باختبار النظام ككل
8-اعادة تشغيل النظام المحدث في بيئة العمل بالمؤسسة المالكة.

الفصل السابع

ضمان الجودة (Quality Assurance)

ضمان جودة البرمجيات

تعني جودة البرمجيات «التوافق مع المتطلبات الوظيفية والأداء المعرفين بوضوح، ومع مَقاييس (جمع مِقْيَاس standard) التطوير الموثقة بوضوح أيضاً، ومع الميزات الضمنية المتوقعة في جميع البرمجيات الاحترافية». لذا تُعتبر المتطلبات البرمجيةُ الأساسَ الذي تقاس عليه الجودة. ويكون القصور في التوافق مع المتطلبات مرادفاً للقصور في الجودة. وتعني ضمان جودة البرمجيات Software Quality Assurance (SQA) توفير المعطيات الكافية بشأن جودة البرمجيات.

ISO 9001 هو مِقيَاس ضمان الجودة المطبق في هندسة البرمجيات. يتضمن هذا المِقيَس 20 متطلباً يجب توفرها لكي يتحقق نظام ضمان الجودة فعلياً.

8-9- ضمان الجودة في تطوير البرمجيات:

8-9-1- النماذج والمعايير

ايزو 17025 هو معيار دولي يحدد المتطلبات العامة لكفاءة لتنفيذ الاختبارات والمعايرة. هناك 15 متطلب إدارة و 10 متطلبات تقنية. هذه المتطلبات تضع الخطوط العريضة لما يجب أن يقوم به مختبر ليصبح معتمد. نظام الإدارة يشير إلى هيكل المنظمة لإدارة عملياتها أو الأنشطة التي تحول المدخلات من الموارد إلى منتج أو خدمة التي تلبى أهداف المنظمة، مثل تلبية متطلبات الجودة للعملاء، الامتثال للأنظمة، أو تحقيق الأهداف البيئية.
الCMMI (نموذج تكامل نضج القدرات) يستخدم بشكل واسع لتطبيق الجودة (PPQA) في منظمة. مستويات النضج لCMMI يمكن تقسيمها إلى 5 خطوات التي يمكن لشركة أن تحققها من خلال أداء أنشطة محددة داخل المنظمة.

8-9-2- ضمان الجودة للشركات

- أهم التقنيات المعتمَدة لضمان الجودة في بناء البرمجيات وتطويرها:

نظراً لأهمية هذه التقنيات المساعدة، واستكمالاً لما بدأنا شرحه في ملف سابق، حول ضمان الجودة وأهميتها لنجاح وإنجاح "مشاريع البرمجيات", وبعد أن تعرضنا بالشرح لكلٍّ من(SEI) و (CMM)، كان وما زال من المهم أيضاً استكمال تناول هذه التقنيات بالشرح والتوضيح ضمن حدود النص المتاح لأهميتها.

- ❖ (ISO : International Organization for Standardization) تخص بالتحديد المعيار (ISO 9001:2000) الذي حل محل المعيار (ISO 1994) المختص بجودة التنظيم الإداري بوجهٍ عام. في معظم الحالات تحتاج المؤسسات إلى تنفيذ التدقيق في الجودة داخلياً (Internal Quality Auditing) استعداداً للخضوع لِتدقيق خارجي (External Quality Auditing) من قبل المؤسسات المخولة بذلك . تقوم هذه المؤسسات بترشيح تلك التي خضعت للتدقيق، لنيل

شهادة الجودة . تستخدم هذه المعايير لضبط الجودة وتحسينها في العديد من مجالات الإدارة، وليس فقط تلك المتخصصة ببناء البرمجيات وتطويرها, وهي تشمل مراحل العمل كلها، بدءاً من التصميم والتوثيق، إلى بناء المنتج، وصولاً إلى فحصه وتركيبه وتقديم الخدمات . تتألف المجموعة الكاملة لهذه المعايير من :

- متطلبات نظام الجودة (9001:2000 Quality Management System / Requirements).
- أسس نظام الجودة (9000:2000 Quality Management System/Fundamentals).
- أدلة نظام الجودة لتحسين الأداء (9004:2000 Quality Management System Guidelines for Performance Improvements).

مدة صلاحية الشهادة تصل إلى ثلاث سنوات ويعاد التقييم بعدها . لكن لا بد من التأكيد هنا أن الحصول على هذه الشهادة لا يضمن بالضرورة جودة المنتج، بل يؤكد فقط أنّ إنتاجه جرى وَفْقَ الإجراءات الموثقة لدى المؤسسة المعنية .

❖ معهد IEEE: (Institute of Electrical and Electronics Engineering) ، يختص هذا المعهد بمجموعة من المعايير مثل :

- معايير توثيق فحص البرمجيات (IEEE Standards for Software Test Documentation).
- معايير فحص وحدات البرمجيات (IEEE-ANSI Standards 1008).
- معايير مخطط ضمان جودة البرمجيات 730 (IEEE-ANSI Standards for Software Quality Assurance Plans 730).

❖ المعهد الوطني الأمريكي للمعايير (ANSI : American National Standards Institute) الذي طور العديد من المعايير المتعلقة بجودة البرمجيات بالتعاون مع (IEEE) و (ASQ American Society for Quality)

□ إن مكتبة البنية الأساسية لتقنية المعلومات (ITIL: Information Technology Infrastructure Library) ، هي مجموعة من المعايير المبنية على أفضل الخبرات في مجال إدارة الخدمات التي تقدمها تقنية المعلومات .وهي مكونة من مجموعة من النصوص المعرفة وفق العمليات الرئيسة التالية :

❖ دعم الخدمات (Service Support)
❖ تقديم الخدمات (Service Delivery)
❖ إدارة الخدمات (Service Management)
❖ الدعم الفني (Software Support)
❖ العمليات الحاسوبية (Computer Operations)
❖ إدارة أمن المعلومات (Security Management)
❖ بيئة العمل (Environmental)

8-11-1- التحقق من نوعية البرمجيات و اختبارها

إن إنفورما ملتزمة بتقديم برمجيات و خدمات تحقق كافة رغبات و تطلعات زبائنها. فهي مزود خدمات معلوماتية و يمكنها أن:

- **تسعى لتحقيق الامتياز في كافة المجالات.**
- **تضمن نوعية ممتازة لبرامجها في جميع مراحل تطويرها.**
- **تحقق أفضل أداء ممكن لكل تطبيقاتها.**

لم تعد النوعية مجرد اتفاق شفهي ،بل أصبحت مجموعة من الشروط المعرّفة جيداً و يجري تطبيقها في جميع مراحل تطوير البرمجية.

8-11-2- المنهجيات و الطرق المستخدمة:

إن وسائل التحقق من نوعية البرمجيات معرّفة تماماً في إنفورما، فإدارة المشاريع، و تنظيم الوقت، و هندسة البرمجيات هي بعض المراحل الأساسية التي نركّز عليها أثناء قيامنا بتطوير البرمجيات و هي تضمن اختباراً دقيقاً و فعالاً لكافة البرمجيات.

8-11-3- اختبار تكامل المعطيات:

يعد اختبار تكامل المعطيات خطوة رئيسية في التحقق من نوعية البرمجيات، وهي تتضمن التأكد من اعتمادية هذه المعطيات و طرق تخزينها للحصول عليها بالشكل المرغوب فيما بعد، و من أنها متماسكة و يمكن الاعتماد عليها ضمن قاعدة البيانات ، يضمن اختبار تكامل المعطيات أيضاً أن المعلومات المطلوبة تخزن ضمن الوقت المحدد، و بالتالي تجنب ضياعها، و الذي يمكن أن يتسبب بأضرار فادحة ضمن الشركات ، يمكن أيضاً التحكم بالبارامترات حسب متطلبات المستخدم ضمن اختبار تكامل المعطيات أثناء تنصيب البرنامج.

8-11-4- اختبار الفعالية و تحليل الأخطاء:

❖ إن المراجعة الدقيقة لمتطلبات الزبون ستساعدنا على اختبار أداء البرمجيات.

تقوم إنفورما بهذه الاختبارات من أجل التوثيق الداخلي للبرامج ، و معرفة الأخطاء التي يمكن أن تظهر أثناء القيام بالاختبارات ، إن تحليل هذه الأخطاء يجب أن يوثّق بشكل جيد كي نزود زبائننا بـ :

- **توصيف كامل للأخطاء و كيفية تجنبها.**
- **تقييم كامل لإمكانية استخدام المنتج و علاقة ذلك بالأهداف المطلوبة.**
- **يتم تضمين البيانات الهامة و المطلوبة من قبل المطورين في الاختبار.**

8-11-5- اختبار الأداء:

يتضمن اختبار الأداء التأكد من تحقيق المنتج للشروط التالية ، ولأي شروط أخرى مطلوبة من قبل الزبون:

1- الاختبار الفيزيائي:

في هذا الاختبار يتم التحقق من البارامترات التالية : التخزين، زمن القيام بالحسابات، إمكانيات التواصل، وزمن استعادة قاعدة المعطيات.

يؤدي ذلك إلى معرفة الحدود القصوى للنظام و مدى استمراريته من أجل تحديد ما هي الجوانب التي يمكن أن يفشل فيها البرنامج أثناء تنفيذه، فالجانب الذي يفشل فيه البرنامج يمكن أن يقود إلى التسبب بأضرار كارثية لكامل البرنامج.

2- اختبار سهولة الاستخدام:

يتضمن هذا الاختبار القيام باختبار اعتمادية الشبكة، و معالج قاعدة البيانات و الذاكرة ... الخ.و هذا أيضاً يتضمن اختبار امكانية دعم المعطيات الأساسية المطلوبة من قبل البرنامج لكي يكون جاهز للعمل في أسوأ الظروف المتوفرة.

3- اختبار الاستجابة:

يستخدم هذا الاختبار لفحص امكانية تحمل النظام لكافة الأعباء الملقاة عليه لقياس زمن الاستجابة.

4- اختبار التوافقية:

يمكن لاختبار التوافقية تحديد المشاكل التي يمكن أن تحدث أثناء ربط المنتج مع البرمجيات و العتاد الصلب. تقوم إنفورما باختبار المنتج عبر مجموعة متنوعة من المنصات و تقييم النتائج التي تسفر عن عن هذا الاختبار.

5- اختبار سهولة الاستخدام:

يمكن عن طريق هذا الاختبار التحقق من مدى سهولة استعمال هذا المنتج من قبل الناس الذين لا يملكون خبرة واسعة في استخدام الحاسوب من أجل مساعدتهم على الاستفادة من كافة الأدوات و الوظائف التي يقدمها هذا المنتج.

هذا يساعد الزبون على الاستفادة من كافة الامكانيات المتاحة ضمن المنتج.

يمكن لهذا الاختبار أن يعطي تحليلاً مفصّلاً لسهولة التنقل بين صفحات المنتج، و الواجهات المستخدمة ... الخ.

إن الاختبار العام لسهولة الاستخدام يتضمن أيضاً القيام بمايلي :

6- فحص واجهات المستخدم الرسومية : و يتضمن فحص واجهات المستخدم الرسومية العناصر التالية :

- **الحقول النصية.**
- **أزرار الخيارات.**
- **أزرار التأشير.**
- **القوائم.**
- **صندوق الاختيار.**

7- نمط اختبار النوعية المفضل: في الصيغة كي نضمن وصول منتجاتنا إلى أعلى المستويات عن طريق التشديد علي اتباع الطرق التالية :-

- **إخضاع الموظفين لبرامج تدريب دورية لرفع مستوى أداءهم.**
- **فحص نوعية البرامج ضمن كافة مراحل تطويرها.**

- **تطوير آليات فعالة للاختبار الداخلي.**
- **تحقيق تواصل فعال ما بين الزبائن و فريق العمل ضمن إنفورما.**

عناصر الجودة: Quality Attributes

شركة (Hewlett-Packard) طورت مجموعة من عناصر جودة البرمجيات والتي اطلق عليها بالاسم المختصر(FURPS)-وهي اخذ الحرف الاول من الكلمات الانجليزية لأسماء العناصر(functionality, usability, reliability, performance, and supportability) وهذه العناصر مثلت هدفا منشودا لكل التصاميم البرمجية:

- الوظيفية : تقاس بتقييم مجموعة خصائص وقدرات البرنامج، بشكل عام هي تمثل كل الوظائف المسلمة والامن والحماية لكامل النظام.
- قابلية الاستخدام (Usability): تقاس باعتبار العوامل البشرية كل الملامح (aesthetics) ، والرصانة والتوثيق.
- الموثوقية (Reliability): تقييم بقياس الذبذبة بين المعاناة من الاخطاء والدقة والنتائج المخرجة معدل الزمن لحصول الاخطاء((MTTF (mean-time-to-failure ، قابلية الاسترجاع من الاخطاء والتوقعات المستقبلية(predictability) عن البرنامج.
- الاداء (Performance): تقاس باعتبارات سرعة المعالجة –وزمن الاستجابة ، واستهلاك المصادر ، والمخرجات والفاعلية.
- المرونة (Supportability): تجمع قابلية التوسع للبرنامج، والتكيف مع التغيرات ، وقابلية الخدمة وهذه يمكن تمثل بمصطلح اعم قابلية الصيانة (maintainability) بالإضافة الى قابلية الاختبار وقابلية النقل وقابلية التنظيم (configurability)والتحكم بعناصر النظام.

تم بحمد الله وتوفيقه،،،