جامعة حمص
الكلية التطبيقية
قسم تقنيات حاسب
السنة الثالثة

هندسة برمجيات
SOFTWARE ENGINEERING
(SE)

د.لودا رشيد علي

فصل ثاني 2025-2026 .

هدف المقرر:

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

Lecture 2

أسس تطوير هندسة البرمجيات

1- مقدمة: تطوير هندسة البرمجيات ليس مجرد كتابة كود، بل هو نهج هندسي منظم يهدف لبناء أنظمة برمجية عالية الجودة، قابلة للصيانة، وفعالة.

يوجد عدد من الأنشطة للتمكن من التعامل مع التعقيد الكبير للأنظمة أهمها : النمذجة **(modeling)**، حل المسائل **(problem solving)**، تحصيل المعرفة **(knowledge acquisition)**.

يمكننا أن نصف هندسة البرمجيات بأنها موجهة بالأسباب **(Rational driven)** فأثناء تحصيل المعرفة وأخذ القرارات المتعلقة بالنظام يجب الاحتفاظ بالظروف والمبررات التي أدت إلى هذه القرارات.

التعبير عن الأسباب التي دفعت إلى قرارات معينة ضمن نماذج خاصة يتيح- في حال طلب تعديلات على النظام ومراجعة القرارات السابقة - معرفة تأثير التعديلات المقترحة.

2- النمذجة: هي عملية تمثيل النظام أو جزء منه بشكل مبسّط ومنظّم، بهدف فهمه وتحليله وتصميمه من خلال نماذج (Models) توضّح: مكونات النظام، العلاقات بينها، طريقة عمل النظام وسلوكه.

تهدف النمذجة إلى: تسهيل فهم النظام المعقّد، تحسين التواصل بين فريق العمل، اكتشاف الأخطاء مبكرًا، وتقليل التكلفة والوقت أثناء التطوير.

تتم عملية النمذجة باستخدام لغة خاصة هي:**UML- Unified Modeling Language**، وفيما يلي أنواع النماذج في UML:

2-1- النماذج الهيكلية (Structural Models): تمثل البنية الثابتة للنظام وتركّز على: "ممّ يتكوّن النظام؟":

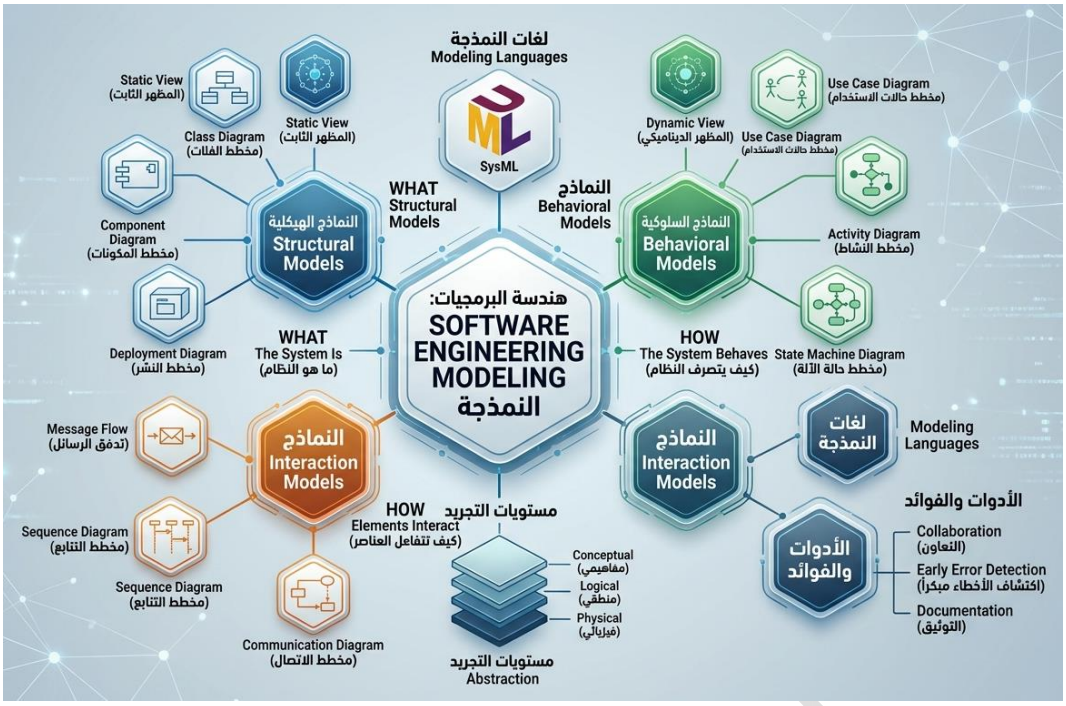
- Class Diagram**(مخطط الفئات)**
- Object Diagram
- Component Diagram

2-2- النماذج السلوكية (Behavioral Models): تمثل سلوك النظام مع الزمن وتركّز على: "كيف يعمل النظام؟":

- Use Case Diagram**(حالات الاستخدام)/(مخطط حالات الاستخدام)**
- Sequence Diagram (تسلسل العمليات)/(مخطط التسلسل)
- Activity Diagram (تدفق العمليات)/(مخطط النشاط)

مثال بسيط في نظام متجر إلكتروني:

- Class Diagram: يوضح (منتج، عميل، طلب)
- Use Case: (شراء منتج، تسجيل الدخول)
- Sequence: خطوات إتمام عملية الشراء



الشكل (2-1)

2-3- مخطط **UCD- Use Case Diagram** : يعرض العلاقة بين الجهات الفاعلة **Actors** وحالات الاستخدام **Use Cases** (مجموعة مهام (خطوات وعمليات متسلسلة) تتم لتحقيق الغرض المطلوب من النظام). يعبر عن الحالة في المخطط بشكل بيضوي يكتب اسمها داخله، يتم استخراجها من نص المسألة عن طريق تحويل الأفعال إلى مصادر.

مثلا: الطالب يستطيع أن يستعير كتاباً ⇦ (استعارة كتاب)

- **Actors** : من يتعامل مع النظام بشكل مباشر دون وسيط

(كل Actor عبارة عن دور ضمن النظام)، وهو فاعل وليس مفعول به، يتم استخراجه من نص المسألة بتتبع الفاعل في الجمل، يأخذ الشكل المجاور و يكتب اسمه تحته.

2-3-1- أنواع الـ Actors:

- **شخص**: موظف- مدير- طالب- طبيب ..(حسب طبيعة النظام)
- **جهاز آلة scanning machine**: يعتمد على فكرة الحساس مثل جهاز الباركود.
- **مؤقت Timer**: يتم ضبطه على موعد محدد فيقوم بتفعيل مهمة معينة كلما حان الوقت مثالها مهمة إرسال إنذار للطلاب المتأخرين عن التسجيل في الجامعة.
- **نظام آخر**: مثل الصراف الآلي ATM.

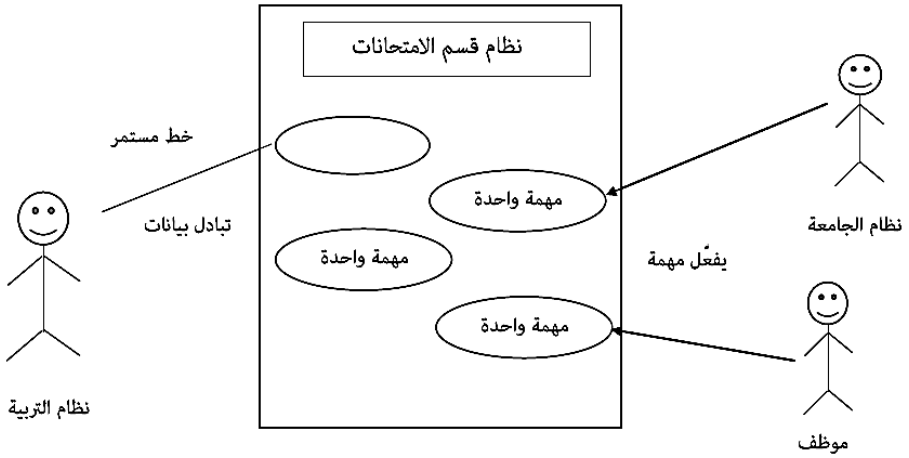
2-3- 2- أصناف الـ Actors:

- **Primary Actor**: من يستخدم النظام بشكل مباشر أي هو الذي يقوم بتفعيل المهام في النظام.
- **Secondary Actor**: يستخدم المهام التي تم تفعيلها.

Business Actor: علاقته بالعمل Business بشكل عام.

عندما يكون الـ Actor (سواء شخص أو نظام آخر) قادر على تفعيل مهمة ما في النظام نضع سهم موجه منه إلى المهمة Use Case ، وعندما يكون الـ Actor نظام آخر لا يستطيع تفعيل مهام في النظام بل يستطيع فقط تبادل بيانات معه فنضع خط مستمر غير موجه بينهما، وعندما يستطيع الـ Actor تفعيل مهام إضافية عن Actor آخر فهو يرث دوره.

مثال:



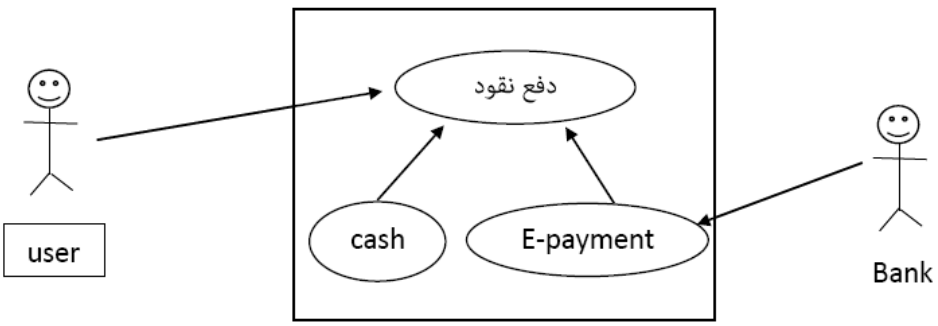
الشكل (2-2) تمثيل نظام الامتحانات

يتم توصيف كل Actor وفق الجدول التالي:

Name	اسم الدور مثلاً رئيس الامتحانات
Type	(primary / secondary)
Description	المهام التي يطلقها " صلاحيات "

2-3-3- العلاقات Relationships بين حالات الاستخدام: يوجد عدة أنواع من العلاقات بين حالات الاستخدام:

العمومية Generalization: مثالها عملية الدفع الإلكتروني:



الشكل (2-3) تمثيل عملية التسديد المصرفية (علاقة عمومية)

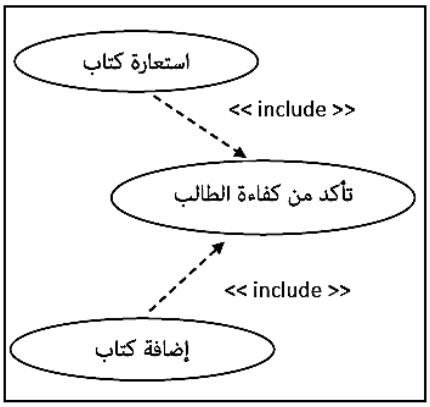
هنا يؤدي الدفع الإلكتروني والكاش نفس المهمة (دفع النقود) ويقوم المستخدم باختيار طريقة الدفع التي تناسبه، يمكن وجود بعض الخطوات المشتركة التي تنفذ في مهمة دفع النقود قبل التفرع إلى الدفع الإلكتروني أو الكاش.

يمكن اعتبار البنك **Secondary Actor** بالنسبة لمهمة دفع النقود.

- **التضمين Include:** عندما تتواجد مجموعة خطوات متسلسلة بأكثر من Use Case أي أنها تتكرر أكثر من مرة، يتم وضعها في Use Case منفصلة وتضمينها في جميع الUse Cases التي تنفذها حيث يتم استدعاؤها بتسلسلها الصحيح ضمن تسلسل الخطوات لكل Use Case.

مثال:

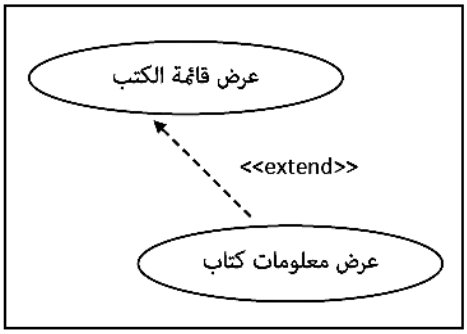
في نظام مكتبة عمليتا استعارة أو إضافة كتاب تحتاجان في مرحلة ما ((التأكد من كفاءة الطالب)) لذلك يتم تضمين مهمة التأكد من الكفاءة في كل منهما برسم خط منقط من حالة الاستخدام المتضمنة إلى حالة الاستخدام المضمنة مكتوب عليها <<include>>.



الشكل (2-4) مثال علاقة التضمين

- الموسعة Extend: الوظيفة الفرعية في هذا النوع قد يستدعيها المستخدم وقد لا يستدعيها وهي خط منقط من المهمة الفرعية إلى الأساسية مكتوب عليها <<extend>>.

مثال:



الشكل (2-5) علاقة موسعة

3- أنواع المنتجات البرمجية:

يوجد حسب الاستخدام نوعان من المنتجات البرمجية قد تطور لزبون محدد أو للتوزيع في الأسواق:

- **برامج عامة (Generic Programs):** برامج منتجة لأغراض عامة يمكن أن يستخدمها أي شخص أو شركة، مثالها:

مجموعة البرامج المكتبية (الأوفيس)- مضادات الفيروسات- مسرع تحميل الإنترنت- منظف الريجستري...وغيرها الكثير.

- **برامج خاصة (Bespoke Programs):** البرامج المعدة خصيصاً لمستخدم محدد **Customized حسب** ما يريد (مستثمر أو شركة)، مثالها:

نظام إدارة كلية- نظام التسجيل- تصميم صفحات وب- نظام مكتبة ...وغيرها العديد.

مقارنة مع البرامج العامة عادة تكون ذات حجم صغير، معقدة الاستخدام، متعبة للمبرمج و ذات إيرادات أقل.

4- العوامل المؤثرة في جودة البرمجيات:

المهام التي تؤديها البرمجيات هي الأساس في درجة الجودة، واعتماد المقاييس العالمية في صناعة البرمجيات يؤدي إلى تصميم برمجيات ذات مواصفات جيدة يمكن أن يعتمد عليها بتحقيق الخصائص المتعارف عليها (أداء وظائفها بكفاءة عالية، توافقها مع أنظمة حاسوب مختلفة، الصيانة الميسرة، القدرة على التطوير والتعديل)، يجب قبل وبعد وأثناء تصميم البرامجيات مراعاة ما يلي:

- **Time الزمن:**

أ -تحديد موعد تسليم النظام.

ب -تحديد الفترة الزمنية للمبرمجين لإنتاج هذا النظام.

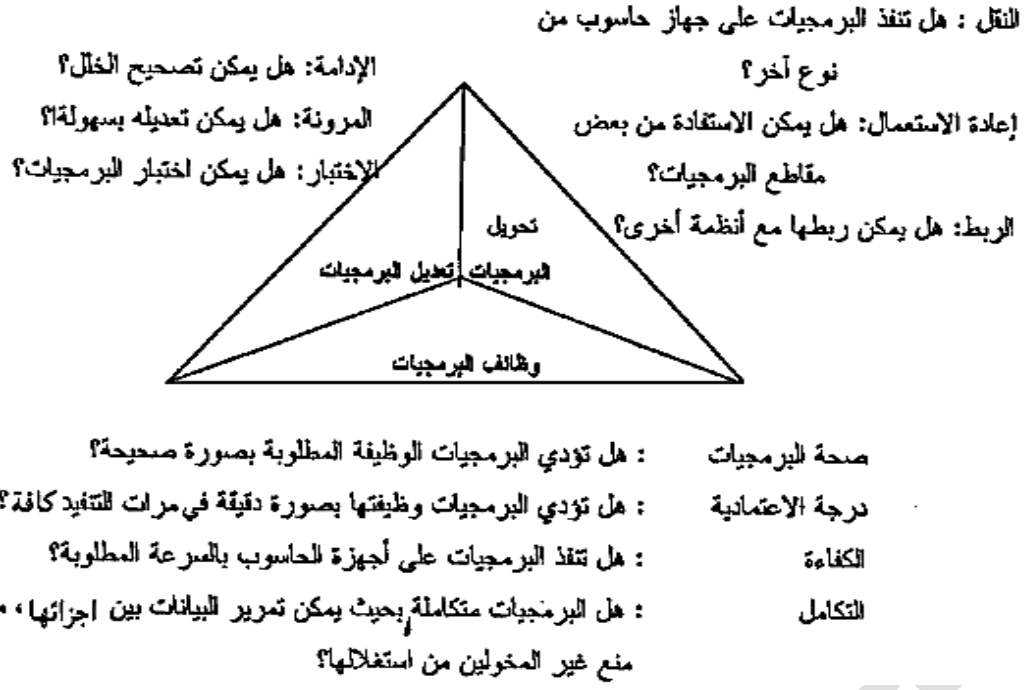
ت -تحديد سرعة النظام أو ما يسمى استغلال موارد النظام.

- **Quality الجودة:** بناء برمجيات تؤدي المهام المطلوبة للجهة المستفيدة منها بكفاءة عالية ضمن المعايير المعتمدة (المقاييس المتبعة) المتعارف عليها بين مراكز صناعة البرمجيات، ويتضح أن:

- البرمجيات **الأقصر** هي الأيسر متابعة.
- البرمجيات ذات **القرارات الأقل** هي الأفضل مع تجنب تداخل القرارات.
- التحديد الجيد لتركيب البيانات.
- الإكثار من **التوضيح والشرح.**
- **الانسجام:** تصميم فقرات البرمجيات بأسلوب موحد ونمطي.
- **التكامل:** يمكن تبادل البيانات بين أجزائها مع ضمان حمايتها وعدم اختراقها.
- **الكفاءة:** استخدام أقل حجم ممكن من الموارد (حجم التخزين، زمن المعالجة).

الفاعلية: قيام البرمجية بالوظائف المطلوبة وفقا لرغبة المستخدم.

يبين الشكل التالي أهم الصفات التي يجب أن تتمتع بها المنتجات البرمجية لتكون جيدة:



الشكل (2-6) جودة المنتجات البرمجية

5- التحديات التي تواجه هندسة البرمجيات:

- **تحدي عدم التجانس** : يزداد الطلب على البرمجيات التي تعمل في بيئة موزعة عبر الشبكات التي تحتوي أنواعًا مختلفة من الحواسيب والأنظمة، فيكمن هذا التحدي في ضرورة تطوير تقنيات تتيح بناء برمجيات تستطيع أن تتماشى مع بيئات التطوير والتنفيذ المختلفة.
- **تحدي التسليم** : يستغرق تطبيق تقانات هندسة البرمجيات زمنًا طويلاً في غالب الأحيان، ومع تسارع إيقاع العمل والتغييرات السريعة المطلوبة، أصبح من الضروري تطوير تقنيات تتيح تسليم البرمجيات بزمن أقل دون التأثير على جودتها.
- **تحدي الثقة:** مع دخول البرمجيات في جميع مناحي الحياة، أصبح من الضروري تطوير تقنيات تثبت للمستخدم أنه يمكنه أن يثق بالبرمجيات.
- **المسؤولية المهنية والأخلاقية:** تتطلب هندسة البرمجيات مسؤولية تتجاوز حدود تطبيق المهارات التقنية، إذ يجب التصرف بأمانة وأخلاق عالية مهنيًا، نقصد بالسلوك الأخلاقي ما هو أكثر من مجرد احترام القانون ويتضمن:

  السرية: يدخل المبرمج إلى شركة الزبون ويطلع على تفاصيل عمله وأسراره المهنية، لذا تقتضي أصول المهنة أن يحفظ هذه الأسرار.

  الكفاءة: تقتضي أصول المهنة بذل قصارى الجهد والعمل وفق السوية المطلوبة، وهذا يعني كذلك عدم الإقدام على عمل يُعرف مسبقًا أنه لا تُمتَلك المهارات الكافية له.

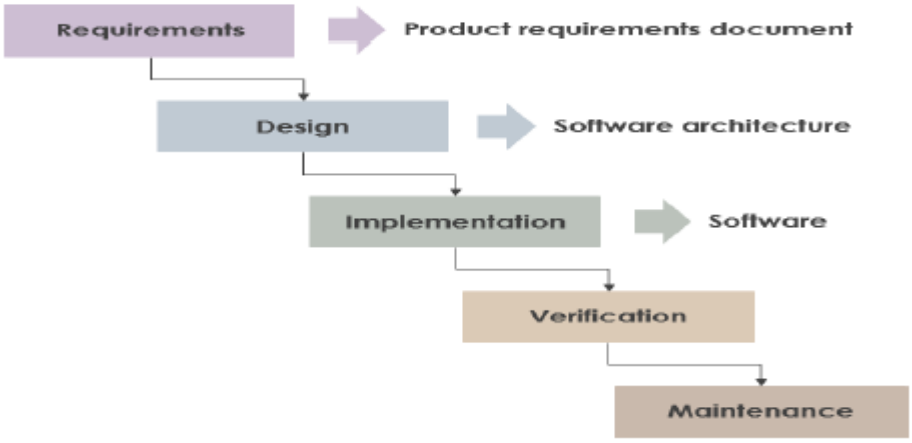
- **حقوق الملكية الفكرية:** تقتضي أصول المهنة الاطلاع على القوانين المحلية التي تحكم استخدام الملكية الفكرية كبراءات الاختراع وحقوق النشر وغيرها، ويجب ضمان حماية الملكية الفكرية للزبائن.
- **عدم إساءة استخدام الحواسيب:** تقتضي أصول المهنة عدم استخدام المهارات التقنية لإساءة استخدام حواسيب الآخرين، وقد تكون الإساءة بسيطة (كاللعب بالألعاب على حواسيب الزبون) أو فادحة (كنشر الفيروسات).

6- الإجرائية البرمجية:

يقصد بها مجموعة الأنشطة من بداية التفكير بالبرمجية مروراً بتحليلها وتصميمها وإنتاجها حتى استخدامها، نعبر عن ذلك بدورة حياة نظام البرمجيات ومراحل تطويره **Software development life cycle (SDLC)** وتقسم إلى المراحل التالية:

- **التحليل (Analysis)**: جمع المعلومات بدقة، **(Requirements Analysis and Definition)** تحديد وتعريف المتطلبات، المهام التي سيقوم بها البرنامج، ودراسة الجدوى، ينتج عن هذه المرحلة مخطط حالات الاستخدام **(Use Case Diagram)** ووثيقة توصيف المتطلبات **SRS (software requirements specification)**.
- **التصميم (Design)**: تحديد كيفية حل المشكلة والدخول في تفاصيل النظام؛ يحدد التصميم هيكلية وبنية النظام من خلال تجزئته إلى أنظمة فرعية وتحديد واجهات المستخدمين والمكونات والوحدات والبيانات بما يحقق متطلبات الزبون؛ ينتج عن هذه المرحلة المخططات والتصاميم.
- **التحقيق (Implementation)**: تحويل المخططات والتصاميم إلى إحدى لغات البرمجة لإنتاج نظام قابل للاستخدام ويلبي احتياجات الزبون؛ تجري في هذه المرحلة بعض الاختبارات على بعض أجزاء النظام للتأكد من عمله بطريقة صحيحة.
- **الاختبار (Testing) أو التحقق (Verification)**: تجميع أجزاء النظام مع بعضها والتأكد من عملها بشكل سليم وفقاً للشروط والمواصفات.
- **الصيانة (Maintenance)**: هي المرحلة الأطول تهدف إلى إبقاء النظام مواكباً للتطورات والمعدات الحديثة؛ جزء من هذه المرحلة يكون في تصحيح الأخطاء والجزء الآخر يكون في تطوير وإضافة تقنيات ووظائف جديدة.

كل مرحلة تتضمن العديد من الخطوات أو النشاطات ولكل منها دخلها وخرجها وتأثيرها على جودة المنتج النهائي (البرنامج).



الشكل (2-7) Software development life cycle

قد تأخذ دورة حياة تطوير البرامج أشكالاً أو أنماطاً أخرى وفق النموذج المتبع في دراسة النظام.

7- نموذج الإجرائية البرمجية:

نموذج الإجرائية البرمجية هو تمثيل مبسط للإجرائية البرمجية يجري عرضه من وجهة نظر معينة، ولفهم المقصود من ذلك نورد الأمثلة التالية:

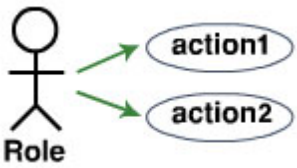
7-1- وجهة نظر تدفق العمل أو سير العمل (**Workflow perspective**) وفيها يمثل النموذج تتالي الأنشطة في الإجرائية مرفقة بالمدخلات والمخرجات التي ترافق هذه الأنشطة.

الشكل (2-8) Workflow perspective

7-2- وجهة نظر تدفق المعطيات (**Data-flow perspective**): وفيها يمثل النموذج تدفق المعلومات وما يطرأ عليها من تحولات بفعل الحواسيب أو المستخدمين.

الشكل (2-9) Data-flow perspective

7-3- وجهة نظر الدور/الفعل (**Role/action perspective**): وفيها يمثل النموذج دور الأشخاص المعنيين بالإجرائية البرمجية والأنشطة التي تقع ضمن مسؤوليتهم.



الشكل (2-10) Role/action perspective

توجد ثلاثة نماذج عمومية تعتمد عليها معظم نماذج الإجرائيات البرمجية، هي:

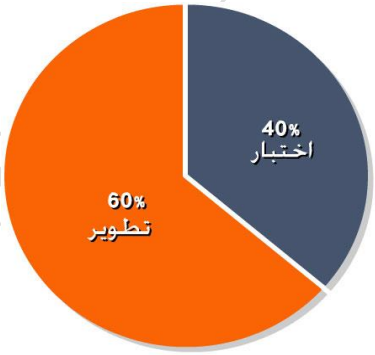
- **المقاربة الشلالية (Waterfall approach)**: تقوم بفصل الأنشطة آنفة الذكر بعضها عن بعض وترتيبها على مراحل متتالية:

توصيف المتطلبات (**Requirements Specification**) ← التصميم (**Design**) ←التحقيق (**Implementation**)←الاختبار (**Testing**).

- **التطوير التكراري (Iterative development)**: تعتمد هذه المقاربة على تداخل أنشطة التوصيف والتصميم والتحقيق والاختبار، وهنا يمكن بناء نظام أولي (ابتدائي) بسرعة باستخدام مواصفات موجزة للنظام المطلوب، يمكن العمل لاحقًا على تحسين النظام بالاستفادة من ملاحظات الزبون على هذا النظام الابتدائي للوصول به إلى الغاية المنشودة، أو يعاد بناء النظام من جديد باستخدام منهجيات أكثر احترافية للحصول على نظام قوي وقابل للصيانة.
- **هندسة البرمجيات المعتمدة على المكونات Component-based SE**: تقوم هذه المقاربة على فرضية أن بعض أجزاء النظام موجودة مسبقًا، وتركز إجرائية التطوير هنا على دمج هذه الأجزاء بدلاً من إعادة بنائها من الصفر.

8- توزع الكلفة المادية **Cost**:

يلاحظ عموماً أن حوالي **60 %** من الكلف تصرف في مرحلة التطوير، و **40 %** في مرحلة الاختبار، أما توزع كلف التطوير على الأنشطة المختلفة فيرتبط بنوعية الإجرائيات المستخدمة وطبيعة البرمجيات قيد التطوير، فأنظمة الزمن الحقيقي تحتاج إلى إجراء عمليات تحقق من الصلاحية واختبار تفوق بكثير تلك المستخدمة في أنظمة الوب، كما أن الخصائص المطلوبة للنظام (كالأداء العالي والموثوقية) تؤثر كثيرًا على هذا التوزع.



الشكل (2-11) توزع كلفة تطوير البرمجية