الجامعـة الافتراضيَّة السوريَّة
SYRIAN VIRTUAL UNIVERSITY

هندسة البرمجيات

الدكتورة غيداء ربداوي

Books

هندسة البرمجيات

الدكتورة غيداء ربداوي

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2018

Software Engineering

Ghaidaa Ribdawi

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2018

الفهرس

الفصل الأول مدخل إلى هندسة البرمجيات

1. مقدمة

من المعلوم أن اقتصاد جميع الدول المتقدمة أصبح اليوم مرتبطاً بالبرمجيات، وقد تزايد عدد الأنظمة التي تتحكم بها البرمجيات في عالمنا تزايداً ملحوظاً. كما أصبحت عائدات البرمجيات تشكل نسبة عالية من الدخل القومي في الدول المتقدمة.

ومع ازدياد الجزء البرمجي في الأنظمة الحديثة، ظهرت عيوب وأخطاء تسببت فيها هذه الأجزاء.

ويمكن الاطلاع على أمثلة عن بعض هذه الأخطاء فيما يلي:

1- خطأ أعوام القرن العشرين **1900**.

في العام 1992 تلقت السيدة ماري من ولاية مينيسوتا دعوة للالتحاق بروضة أطفال وهي تبلغ من العمر 104 سنة!

2- خطأ السنين الكبيسة.

غُرم أحد المتاجر بمبلغ 1000$ بسبب احتفاظه باللحم يوماً إضافياً في 29 شباط 1988. ويعود الخطأ في ذلك إلى البرنامج الذي استخدم لطباعة تاريخ انتهاء الصلاحية الذي لم يتنبه إلى أن العام 1988 هو سنة كبيسة.

3- تصميم سيء للواجهات.

في نيسان 1990 غادر قطار في لندن المحطة دون أن يكون السائق على متنه. فقد ضغط السائق زر الانطلاق لكن البرنامج كان مصمماً بحيث لا يتحرك القطار حتى تكون جميع أبوابه مغلقة. كان أحد الأبواب عالقاً فنزل السائق ليحرره وما إن أغلق الباب حتى انطلق القطار دون سائقه.

4- الأمن.

ارتفع عدد الحوادث الأمنية التي أُبلِغ بها **CERT**[1] من 252 حادثاً في 1990 إلى 21756 حادثاً في 2000 وبلغ أكثر من 40000 حادثاً في 2001.

5- التسليم المتأخر وتجاوز الميزانية.

في العام 1995 تسببت الأخطاء الموجودة في نظام شحن الحقائب المعد لمطار دنفر الدولي الجديد بإتلاف الحقائب الصغيرة. أدى هذا الخطأ إلى تأخير افتتاح المطار لمدة 16 شهراً وإلى زيادة في الكلفة بلغت 3.2 مليار دولار وفي النهاية كانت معظم مراحل نظام شحن الحقائب يدوية.

6- التسليم ضمن المدة.

بعد 18 شهراً من التطوير تم تسليم نظام لشركة تأمين صحي في ويسكنسون بلغت كلفته 200 مليون دولار. غير أن النظام لم يكن يعمل بشكل صحيح! احتاج تعديله إلى دفع كلف وصلت إلى 60 مليون دولار واستغرق الأمر ثلاث سنوات!

7- التعقيد غير الضروري.

كلفت طائرة شحن البضائع **C-17** أكثر من 500 مليون دولار فوق ميزانيتها الأصلية بسبب أخطاء في النظام البرمجي للطيران. في الواقع تحمل هذه الطائرة على متنها 19 حاسوباً و80 معالجاً صغرياً وكتبت أنظمتها بست لغات برمجة مختلفة!

نتجت جميع الأخطاء الآنفة الذكر عن أخطاء في البرمجة. في الحقيقة تعتبر البرمجيات مصنوعات بشرية ذات **تعقيد** عالٍ، فهي تقوم بوظائف كثيرة، ويطلب منها أن تحقق أهدافاً قد تكون في كثير من الأحيان متعارضة. كما أنها تتألف من مكونات عديدة معقدة في حد ذاتها. هذا عدا عن أن المشاريع البرمجية تتعرض أثناء مراحل التطوير لكثير من **التغيير** في المتطلبات بسبب تطور الاحتياجات وتطور السوق.

أما من حيث الكلف، فلو تأملت أسعار الأنظمة الحالية، لوجدت أن أسعار البرمجيات قد تجاوزت بكثير أسعار التجهيزات. ولو تابعت عملية التطوير البرمجي عن كثب، للاحظت أن الكلف المدفوعة في صيانة البرمجيات تفوق كلف بنائها، حتى أن الدراسات قد بينت أن كلف الصيانة يمكن أن تصل إلى عدة أضعاف كلف البناء في الأنظمة التي تستمر في العمل (تعيش) لسنوات طويلة.

من الطبيعي إذاً أن نمعن النظر في العلم الذي يعنى ببناء هذه البرمجيات والذي يسمى **هندسة البرمجيات**، وما يحتويه من نظريات ومنهجيات وأدوات للمحترفين.

2. أسس هندسة البرمجيات

تقوم هندسة البرمجيات على ممارسة عدد من الأنشطة.

1- وتعتبر النمذجة modeling[2] أحد هذه الأنشطة الهامة. يلجأ مهندسو البرمجيات إلى النمذجة ليتمكنوا من التعامل مع التعقيد الكبير للأنظمة.

2- كما تقوم هندسة البرمجيات على ممارسة نشاط آخر هو **حل المسائل problem solving** ، إذ تستخدم النماذج التي توضع لإيجاد حل مقبول للمسألة المطروحة. يجري اختيار الحل من عدة خيارات بناء على خبرات المهندسين المكتسبة وتوجه الشروط المحدودة للموارد (الكلف والزمن) هذا الاختيار.

[1] مؤسسة حكومية تدعم الأفراد عند حدوث حوادث تتعلق بالأمن وتقدم لهم المعارف اللازمة المتعلقة بالأمن، وهو يتبع معهد هندسة البرمجيات.

3- إضافة إلى النشاطين السابقين، تعتمد هندسة البرمجيات على ممارسة نشاط آخر هو **تحصيل المعرفة knowledge acquisition**، فأثناء عملية نمذجة فضاء المسألة والحل يقوم المهندسون بجمع المعطيات، ثم يقومون بتجميعها وتنظيمها للحصول على المعلومات، ومن ثم تشكيلها ضمن معارف. ولا تتم عملية تحصيل المعرفة دفعة واحدة وعلى التتالي، فقد تؤدي معلومة لم تكن بالحسبان إلى هدم النموذج الموضوع كاملاً وإلى إعادة بنائه.

وأخيراً يمكننا أن نصف هندسة البرمجيات بأنها **موجهة بالأسباب rationale driven** . فأثناء تحصيل المعرفة وأخذ القرارات المتعلقة بالنظام، يجب أن يحتفظ مهندس البرمجيات بالظروف والمبررات التي أدت به إلى هذه القرارات. إن التعبير عن هذه الأسباب التي دفعت إلى قرارات معينة ضمن نماذج خاصة يسمح في حال طلب تعديلات على النظام ومراجعة القرارات السابقة بمعرفة تأثير التعديلات المقترحة.

3. الأسئلة الأكثر انتشاراً في هندسة البرمجيات

مع انتشار البرمجيات وافتتاح اختصاصات جامعية في الكليات تعنى بتعليم العلوم التقنية الحديثة، كثرت الأسئلة والاستفسارات حول بعض المفاهيم والفروق الدقيقة فيما بينها، وإليك بعضاً من هذه الأسئلة:

1- ماذا نعني بالبرمجيات software؟
2- ماذا نعني بهندسة البرمجيات Software Engineering؟
3- ما الفرق بين هندسة البرمجيات Software Engineering وعلم الحاسوب computer science؟
4- ما الفرق بين هندسة البرمجيات Software Engineering وهندسة النظم System Engineering؟
5- ماذا نعني بالإجرائية البرمجية software process؟
6- ماذا نعني بنموذج الإجرائية البرمجية software process model؟
7- كيف تتوزع الكلف costs عند هندسة البرمجيات؟
8- ما هي طرائق هندسة البرمجيات software engineering methods؟
9- ما هي الأدوات المساعدة في هندسة البرمجيات CASE (Computer-Aided Software Engineering)؟
10- ما هي خصائص البرمجيات الجيدة attributes of good software؟
11 ما هي أهم التحديات التي تواجه هندسة البرمجيات؟

[2] النمذجة هي عملية تقتضي التركيز في مرحلة معينة على التفاصيل التي تهمنا وتجاهل جميع التفاصيل الأخرى. وأثناء تطوير النظم البرمجية، يبني مهندسو البرمجيات عدداً كبيراً من النماذج models المختلفة للنظام ولمجال التطبيق.

سنرد في الفقرات التالية عن هذه الأسئلة لعل الصورة تصبح أوضح.

1 ماذا نعني بالبرمجيات software؟

تستعمل كلمة برمجية أو برمجيات للدلالة على أحد المفاهيم التالية:

1- البرامج الحاسوبية والوثائق المرافقة لها مثل المتطلبات والنماذج التصميمية وأدلة المستخدم.

2- المنتجات البرمجية التي قد **تطور**[3] لزبون محدد أو للتوزيع في الأسواق. تلاحظ هنا أننا نميز بين نوعين من المنتجات:

عمومية: بمعنى أنها قد طورت لتناسب احتياجات العديد من الزبائن المختلفين، ونضرب مثالاً عنها البرمجيات المطورة للحواسيب الشخصية مثل Excel أو Word.

مفصلة: بمعنى أنها قد طورت لتناسب احتياجات زبون وحيد بناء على المتطلبات التي يضعها.

ومن الجدير بالذكر هنا أنه يجري بناء البرمجيات الجديدة عبر بناء برامج جديدة أو تشكيل (Configure) نظم برمجية عمومية أو إعادة استخدام برمجيات موجودة.

2 ماذا نعني بهندسة البرمجيات Software Engineering؟

هندسة البرمجيات هي الاختصاص الهندسي الذي يُعنى بجميع الجوانب المتعلقة بإنتاج البرمجيات. من الناحية التقنية يعرض هذا الاختصاص مجموعة التقنيات والمنهجيات والأدوات التي تساعد على بناء أنظمة برمجية عالية الجودة ضمن ميزانية معطاة وخلال فترة زمنية محددة، وذلك رغم التطورات الحاصلة باستمرار وأثناء التطوير.

3 ما الفرق بين هندسة البرمجيات Software Engineering وعلم الحاسوب computer science؟

يهتم علم الحاسوب بدراسة النظريات والمنهجيات التي تشكل الأساس في بناء النظم الحاسوبية والبرمجية. أما هندسة البرمجيات فتركز على الجوانب العملية المتعلقة ببناء البرمجيات المفيدة وتسليمها. إن الاطلاع على بعض المعارف المتعلقة بعلم الحاسوب مفيد لمهندس البرمجيات بنفس المقياس الذي نجد فيه أن الاطلاع على بعض المعارف المتعلقة بالفيزياء مفيد لمهندس الإلكترون. ومع ذلك فإن نظريات علم الحاسوب لا تستطيع دوماً أن تعطي الحل لبعض المسائل المعقدة التي تحتاج إلى حل برمجي، وهنا يتم اللجوء إلى مقاربات "حسب الحالة" أثناء تطوير البرمجيات

[3] إن كلمة تطوير هنا هي ترجمة للكلمة الإنكليزية development التي تعني إنشاء وبناء البرمجيات، وستستخدم في هذا المقرر بهذا المعنى وليس بمعنى إجراء تطوير على شيء موجود سابقاً كما قد يخطر ببالك.

4- ما الفرق بين هندسة البرمجيات software engineering وهندسة النظم System engineering ؟

تهتم **هندسة النظم** بجميع الجوانب المتعلقة بتطوير النظم المعتمدة على الحاسـوب computer-based systems بما فيها من تصميم وبناء للتجهيزات والبرمجيات ومن عناية بإجرائية التطوير. أما **هندسة البرمجيات** فهي ذلك الجزء من هذه الإجرائية الذي يهتم بتطوير البنية البرمجية الأساسية والتحكم والتطبيقات وبنى المعطيات التي تدخل في تكوين هذا النظام. وتعتبر هندسة النظم أقدم من هندسة البرمجيات فقد تمكن المهندسون وخلال الأعوام المائة الماضية من بناء وتجميع العديد من الأنظمة الضخمة كالطائرات والمعامل. ولكن مع تزايد نسبة البرمجيات في الأنظمة أصبح من الضروري الاهتمام بهذا الجزء بشكل خاص. وتقع مسؤولية تعريف المتطلبات وتصميم البنيان العام والتكامل ونشر كامل النظام بأجزائه البرمجية وبتجهيزاته على مهندس النظم.

5- ماذا نعني بالإجرائية البرمجية Software process؟

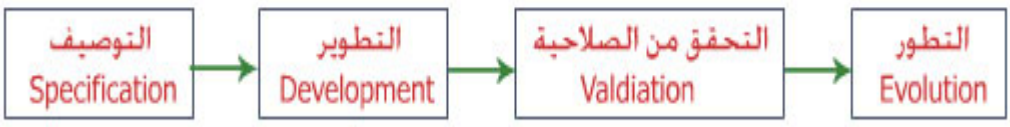
الإجرائية البرمجية هي مجموعة الأنشطة التي تهدف إلى تطوير البرمجيات وتطورها. ومهما اختلف المختصون على تفاصيل هذه الأنشطة، فإنهم متفقون على وجود الأنشطة العمومية الأربعة التالية:

1- **التوصيف Specification:** ويهدف إلى تحديد ما يجب على النظام فعله وشروط التطوير.

2- **التطوير Development:** ويهدف إلى إنتاج النظام البرمجي.

3- **التحقق من الصلاحية Validation**: وتهدف إلى التأكد من أن البرمجية الناتجة هي فعلاً الشيء الذي كان المستخدم ينشده.

4 **التطور Evolution:** ويعنى بالتعديلات على البرمجية استجابة للتغيرات في المتطلبات.



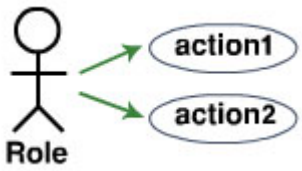
6 ماذا نعني بنموذج الإجرائية البرمجية Software process model؟

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

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

2- وجهة نظر تدفق المعطيات Data-flow perspective: وفيها يمثل النموذج تدفق المعلومات وما يطرأ عليها من تحولات بفعل الحواسيب أو البشر.

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



وتوجد ثلاثة نماذج عمومية[4] تعتمد عليها معظم نماذج الإجرائيات البرمجية، وهي:

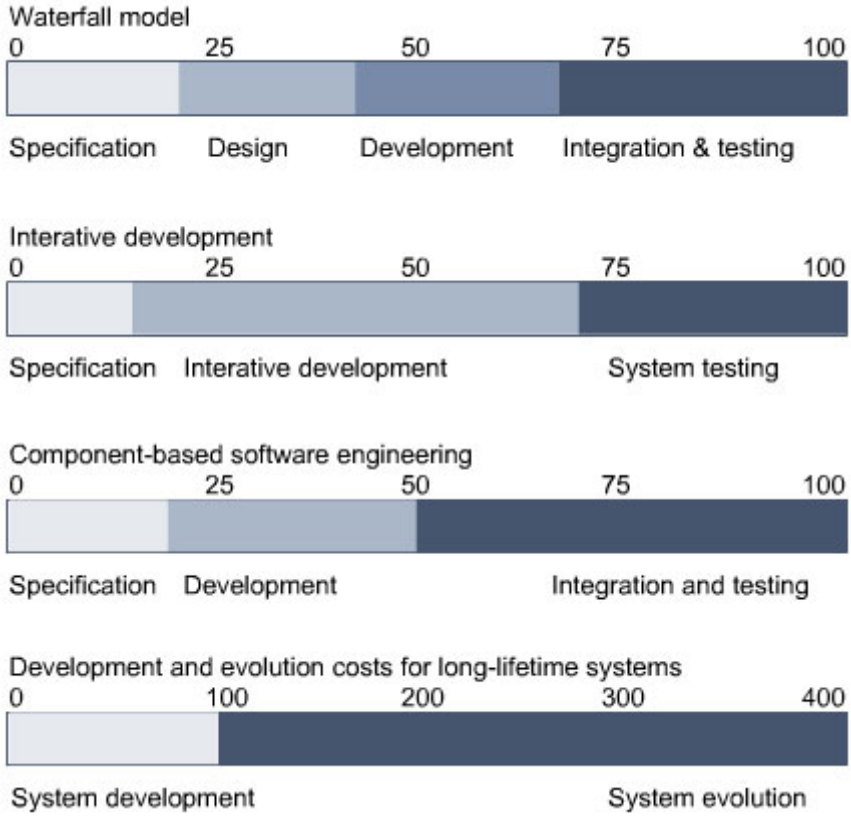
1- **المقاربة الشلالية Waterfall approach**: تقوم بفصل الأنشطة آنفة الذكر بعضها عن بعض وترتيبها على مراحل متتالية، فتظهر لدينا المراحل التالية: توصيف المتطلبات requirements specification والتصميم design والتنجيز implementation والاختبار testing.

2- **التطوير التكراري Iterative development:** تعتمد هذه المقاربة على تداخل أنشطة التوصيف والتصميم والتنجيز والاختبار. وهنا يمكن بناء نظام أول (ابتدائي) بسرعة باستخدام مواصفات موجزة للنظام المطلوب. يمكن العمل لاحقاً 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 صلب البنية وقابل للصيانة.

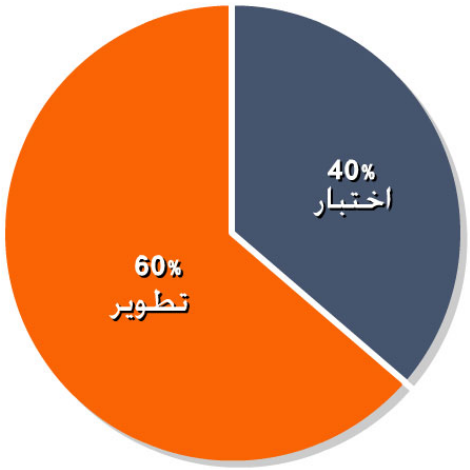
3- **هندسة البرمجيات المعتمدة على المكونات Component-based software engineering:** تقوم هذه المقاربة على فرضية أن بعض أجزاء النظام موجودة مسبقاً. وتركز إجرائية التطوير هنا على دمج هذه الأجزاء بدلاً من إعادة بنائها من الصفر.

7- كيف تتوزع الكلف costs عند هندسة البرمجيات؟

نلاحظ على وجه الإجمال أن حوالي 60% من الكلف تصرف في مرحلة التطوير، وتصرف 40% من الكلف في مرحلة الاختبار. أما كيف تتوزع كلف التطوير على الأنشطة المختلفة، فيصعب إيجاد إجابة بسيطة على هذا السؤال نظراً لاختلاف 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 كثيراً على هذا التوزع. يمكننا في الشكل التالي ملاحظة اختلاف التوزع باختلاف المقاربة والإجرائية المعتمدة.



كما نلاحظ أنه في حالة الأنظمة المفصلة لزبون محدد، تفوق الكلف اللازمة للتطور تلك التي تصرف عند بناء (تطوير) البرمجية.



8- ما هي طرائق هندسة البرمجيات software engineering methods؟

في هندسة البرمجيات نطلق اسم طريقة software engineering method على مجموع نماذج النظام وطرق التدوين والقواعد والنصائح التصميمية وتوجيهات الإجرائية التي تعتمدها مقاربة منهجية ما في تطوير البرمجيات. توجد العديد من الطرائق في هندسة البرمجيات ففي السبعينيات من القرن الماضي ظهرت الطرائق الموجهة بالوظيفة function oriented methods مثل طريقة **التحليل البنيوي Structured Analysis** التي وضعها DeMarco وطريقة **JSD** التي وضعها Jackson ، وفي الثمانينيات والتسعينيات من القرن نفسه ظهرت الطرائق الموجهة بالأغراض مثل طريقة Booch و Rumbaugh. ولكل طريقة من هذه الطرق فلسفتها وتوصيفها ونماذجها المعتمدة، ومخططاتها البيانية وقواعدها وتوجيهاتها وترتيبها الخاص بها لأنشطة التطوير.

9- ماهي الأدوات المساعدة في هندسة البرمجيات CASE (Computer-Aided Software Engineering)؟

أنشئت الكلمة CASE من الأحرف الأولى لكلمات العبارة Computer-Aided Software Engineering ويندرج تحت هذه الكلمة عدد كبير من البرامج المستخدمة لدعم أنشطة إجرائية تطوير البرمجيات. ومن المؤكد أنك استخدمت بعضاً منها فمحررات البرامج وبيئات التفلية debugging هي من هذه الأدوات.

في الواقع يوجد لكل طريقة من طرائق هندسة البرمجيات أنظمة CASE تدعمها. وتقسم هذه الأنظمة إلى قسمين:

1- **أنظمة CASE التي تهتم بالمراحل العليا من التطوير (Upper-CASE):** وهي الأدوات التي تدعم الأنشطة الأولى في إجرائية التطوير أي التوصيف والتصميم.

2- **أنظمة CASE التي تهتم بالمراحل الدنيا من التطوير (Lower-CASE):** وهي الأدوات التي تدعم الأنشطة الأخيرة في إجرائية التطوير أي البرمجة والتفلية والاختبار.

10- ما هي خصائص البرمجيات الجيدة attributes of good software؟

من المؤكد أن بناء البرمجيات عملية معقدة، تنتهي بتسليم الزبون البرمجية التي يتوقعها. ويفترض أن تؤدي البرمجية الوظائف التي طلبها الزبون. لكن يمكن لعدة مهندسين تطوير البرمجية المطلوبة مع وجود اختلاف كبير في الجودة فيما بين منتجاتهم. فما هي الخصائص التي تحدد جودة البرمجيات؟ يوجد العديد من الخصائص التي تسمح بتقييم جودة البرمجية وهي تسمى أيضاً بالمواصفات غير الوظيفية; وهي خصائص هامة قد تؤثر في قبول البرمجية والتفاعل معها أو رفضها. تختلف هذه الخصائص الحرجة من نظام لآخر ففي الأنظمة المصرفية يعتبر الأمان هو معيار الجودة، في حين

تكون درجة الموثوقية في مقسم هاتفي هي المعيار الأهم، أما في الألعاب التفاعلية فإن سرعة استجابة النظام هي الأهم.

وقد جرى تعريف عدد من الخصائص تعتبر مميزات النظام البرمجي الجيد يمكن تعدادها فيما يلي:

1- **الصيانية Maintainability**: يجب أن تبنى البرمجية بحيث تكون قابلة للتطور مع تغير الاحتياجات وهو أمر يعد حتمياً في بيئة الأعمال اليوم.

2- **الاعتماديـة Dependability**: تتضمن الاعتمادية عدداً من الخصائص كالموثوقية، والأمن والأمان. وتكون البرمجية قابلة للاعتماد عليها إذا كان حدوث عطل فيها لا يتسبب بأذى اقتصادي أو فيزيائي.

3- **الفعالية Efficiency**: يجب أن تبنى البرمجية بحيث لا تفرط في موارد النظام كالذاكرة أو دورات المعالج بل تستخدمها بفعالية.

4- **مدى القبول Acceptability**:وهي تعكس مدى قبول المستخدم للبرمجية التي صممت له وهنا يجب أن تكون البرمجية سهلة الاستخدام والفهم وقابلة للتكامل مع النظم الأخرى.

11- ما هي أهم التحديات التي تواجه هندسة البرمجيات؟

تكمن أهم التحديات التي تواجه هندسة البرمجيات في القرن الواحد والعشرين في النقاط الثلاث التالية:عدم التجانس والتسليم والثقة.

1- **تحدي عدم التجانس:** يزداد الطلب على البرمجيات التي تعمل في بيئة موزعة عبر الشبكات التي تحتوي أنواعاً مختلفة من الحواسيب والأنظمة. يكمن هذا التحدي في ضرورة تطوير تقنيات تتيح بناء برمجيات تستطيع أن تتماشى مع بيئات التطوير والتنفيذ المختلفة.

2- **تحدي التسليم**: يستغرق تطبيق تقانات هندسة البرمجيات زمناً 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.

3- **تحدي الثقة:** مع دخول البرمجيات في جميع مناحي الحياة، أصبح من الضروري تطوير تقنيات تثبت للمستخدم أنه يمكنه أن يثق بالبرمجيات.

4. المسؤولية المهنية والأخلاقية

تتطلب هندسة البرمجيات مسؤولية تتجاوز حدود تطبيق المهارات التقنية، إذ يجب أن يتصرف مهندس البرمجيات بأمانة وخلق عالٍ إذا كان يريد أن يكون محترماً مهنياً. ونقصد بالسلوك الأخلاقي ما هو أكثر من مجرد احترام القانون. فما هي أهم أصول المهنة؟

1- **السرية.**

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

2- **الكفاءة.**

تقتضي أصول المهنة أن يبذل مهندس البرمجيات قصارى جهده وأن يعمل وفق السوية المطلوبة. وهذا يعني كذلك أن لا يقدم على عمل يعرف مسبقاً أنه لا يملك المهارات الكافية له.

3- **حقوق الملكية الفكرية.**

تقتضي أصول المهنة أن يكون المهندس على اطلاع على القوانين المحلية التي تحكم استخدام الملكية الفكرية كبراءات الاختراع وحقوق النشر، وغيرها. ويجب أن يضمن المهندس حماية الملكية الفكرية لزبائنه.

4- **عدم إساءة استخدام الحواسيب.**

تقتضي أصول المهنة أن لا يستخدم مهندس البرمجيات مهاراته التقنية لإساءة استخدام حواسيب الآخرين. وقد تكون الإساءة بسيطة (كاللعب بالألعاب على حواسيب الزبون) أو فادحة (كنشر الفيروسات).

وقد قامت مؤسستا **ACM** و **IEEE** بوضع القواعد الأخلاقية لممارسة المهنة. ويمكن الاطلاع عليها في الموقع www.acm.org/about/se-code. تتضمن هذه القواعد ثمانية بنود تتعلق بسلوك مهندس البرمجيات المحترف وبالقرارات التي عليه أن يأخذها. تتجه هذه البنود لجميع المعنيين بهندسة البرمجيات من ممارسين ومعلمين ومدراء ومشرفين وصانعي سياسات ومتدربين وطلاب.

5. الإجرائية البرمجية

1- النماذج العمومية للإجرائية البرمجية

سبق أن ذكرنا في الفقرة 3.6. أنه يوجد ثلاثة نماذج للإجرائية البرمجية وهي:

النموذج الشلالي waterfall model

يتألف هذا النموذج من المراحل التالية:

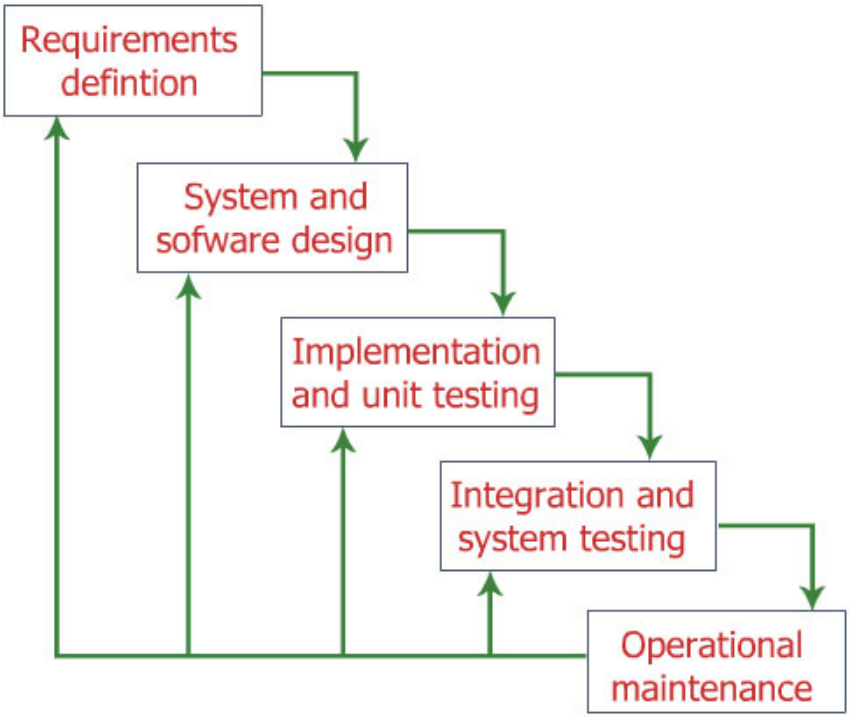
1- تحليل المتطلبات وتعريفها.

2- تصميم النظام والبرمجية.

3- التنجيز والاختبارات الأحادية.

4- التكامل واختبار النظام.

5- التشغيل والصيانة.



إن الانتقاد الرئيسي الموجه لهذا النموذج هو صعوبة أخذ التغييرات المطلوبة بعين الاعتبار بعد أن تقلع الإجرائية، وذلك لأننا يجب أن ننتهي تماماً من مرحلة ما قبل الانتقال إلى المرحلة التالية. لهذا السبب، يناسب هذا النموذج الأنظمة التي تكون متطلباتها واضحة جداً والتغييرات المتوقعة على هذه المتطلبات محدودة أثناء مرحلة التصميم.

يستخدم هذا النموذج في المشاريع الهندسية المتعلقة بالأنظمة الكبيرة حيث يجب بناء النظام في مواقع متعددة.

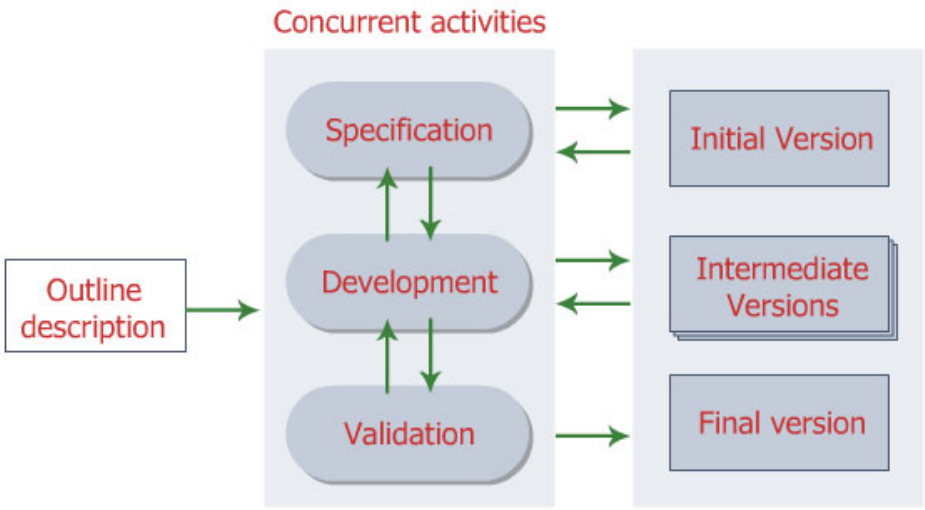
يمكن أن يستخدم هذا النموذج في حالتين:

1- **حالة التطوير الاستكشافي Exploratory development**.

في هذه الحالة يبدأ العمل مع الزبون انطلاقاً من توصيف للخطوط العريضة للنظام بهدف التطور تدريجياً للوصول إلى النظام النهائي. وتكون البداية عادة مع متطلبات واضحة ومفهومة، ثم يضاف إلى النظام الخصائص الجديدة عندما يقترحها الزبون.

2- **حالة النمذجة الأولية throw-away prototyping**.

تحدث هذه الحالة عندما يكون الزبون غير قادر على توصيف النظام، فيجري بناء نموذج أولي يتفق من خلاله على المواصفات المطلوبة للنظام. وبعد فهم المواصفات يُرمى النموذج الأولي ثم يبنى النظام.



النموذج التطوري evolutionary model

إن الانتقادات الرئيسية الموجهة لهذا النموذج تكمن في عدم وضوح الإجرائية ومراحلها، وفي الحصول على نظام ذي بنية مخلخلة، إضافة إلى أن بناء النموذج الأولي يتطلب مهارات وإتقان لغات خاصة (مثل Visual Basic).

يناسب هذا النموذج الأنظمة التفاعلية الصغيرة أو متوسطة الحجم، كما يمكن أن يستخدم في بعض أجزاء الأنظمة الكبيرة كالجزء المتعلق ببناء واجهات المستخدم، أو في النظم التي تكون مدة حياتها قصيرة.

هندسة البرمجيات المعتمدة على المكونات components based software Engineering

تعمل هذه الإجرائية وفق المراحل التالية:

1- توصيف المتطلبات اللازمة للنظام.

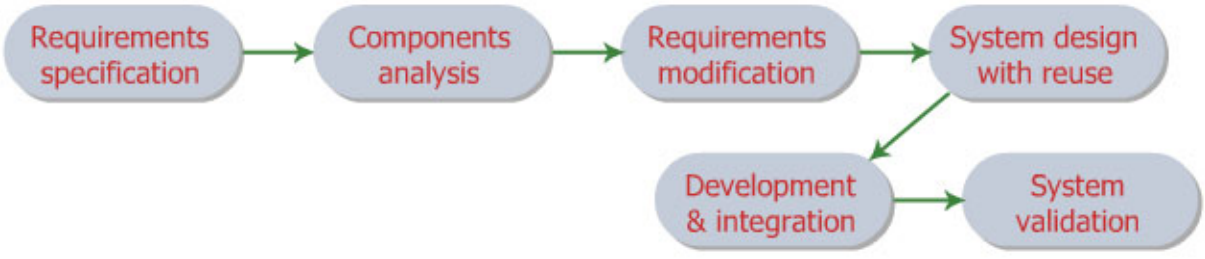
2- تحليل المكونات الموجودة في السوق ومدى تحقيقها للمتطلبات الخاصة بالنظام.

3- تعديل المتطلبات بما يتوافق مع المكونات الموجودة[5].

4- تصميم النظام مع إعادة استخدام المكونات الموجودة.

5- التطوير والتكامل بين المكونات.

6- التحقق من صلاحية النظام.



[5] ينسق المهندس مع الزبون لضمان موافقته على التعديلات.

سبق وأشرنا إلى أننا في هذا النموذج من الإجرائية البرمجية نقوم بعملية " إعادة استخدام reuse " لمكونات جاهزة COTS (Commercial-off-the-shelf) يجري إجراء التكامل في بينها لبناء النظام.

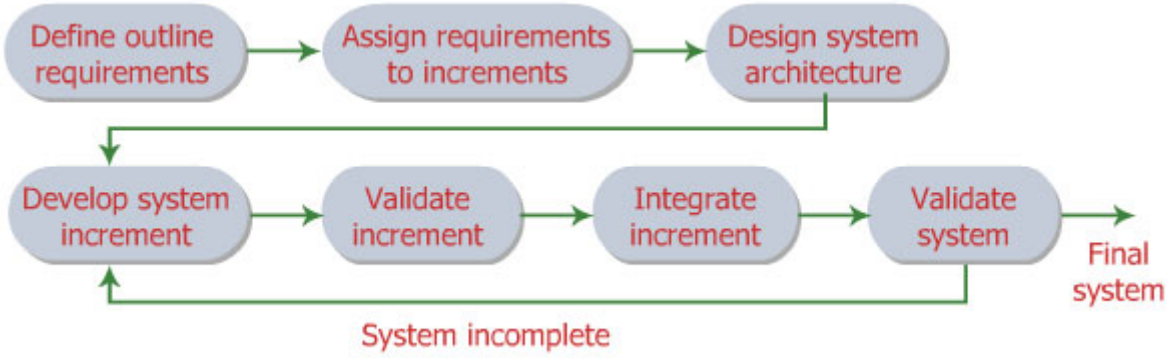
2- التكرار في الإجرائية

تتطور متطلبات الأنظمة تطوراً مستمراً أثناء سير العمل في المشروع، وعليه فإن التكرار يعتبر جزءاً لا يتجزأ من الإجرائية، حيث علينا تكرار العمل على المراحل السابقة دوماً في الأنظمة الكبيرة.

ويطبق هذا التكرار على أي نموذج من النماذج العمومية للإجرائية. يمكن إجراء هذا التكرار باستخدام إحدى المقاربتين التاليتين:

التسليم التزايدي incremental delivery

بدلاً من تسليم النظام دفعة واحدة، يُجزأ التطوير والتسليم إلى تزايدات increments حيث يؤمن كل تزايد جزءاً من الوظائف المطلوبة. لتحقيق هذا الأمر في الإجرائية، تُرتب متطلبات المستخدم حسب الأولويات التي يحددها المستخدم، ثم يبدأ العمل على المتطلبات ذات الأولويات الأعلى. وبمجرد أن يبدأ تطوير التزايد الذي يحقق هذه المتطلبات "تُجمد" المتطلبات المتعلقة به في حين يبقى من الممكن تغيير المتطلبات المتعلقة بالتزايدات التالية.

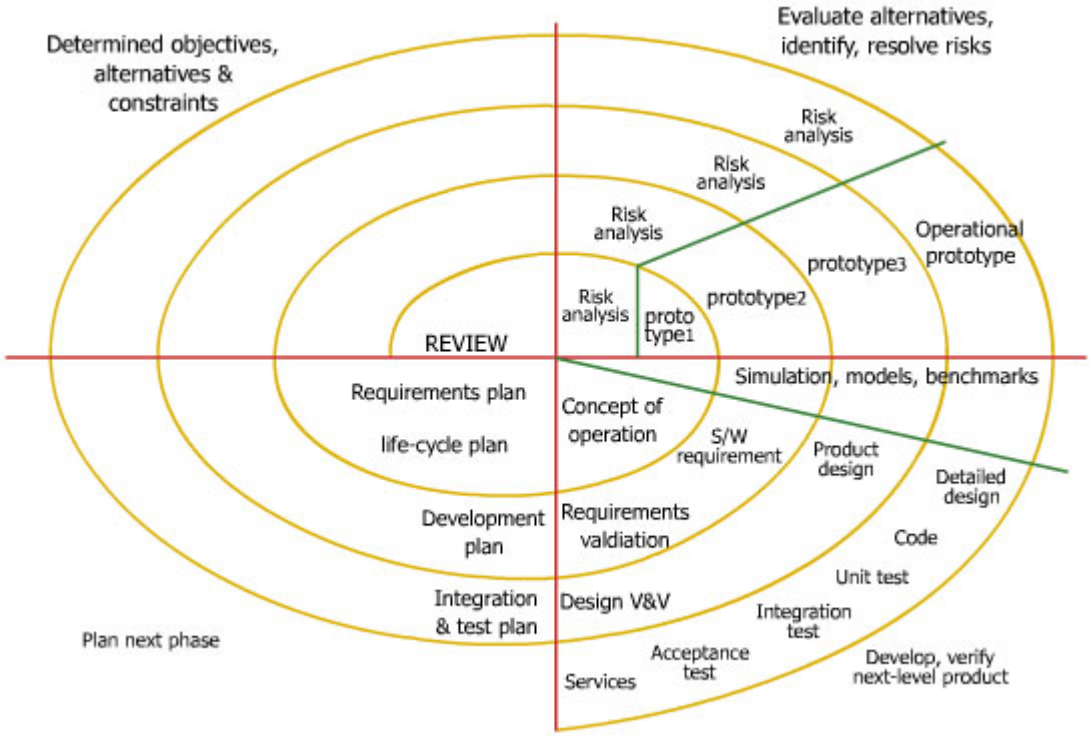


يتمتع هذا التطوير التزايدي بميزات عدة، إذ يبدأ العمل على أكثر ما يهم المستخدم وهكذا يحصل في وقت مبكر على إصدار من النظام يلبي حاجاته الملحة. كما أن كل تزايد يعتبر بمثابة "نموذج أولي" prototype يسمح بإيضاح المطلوب للتزايدات اللاحقة.

إن العمل وفق هذه الآلية يحمي النظام من الفشل، لاسيما وأن الخدمات ذات الأولوية العالية تحظى بالنصيب الأكبر من الاختبار.

التطوير الحلزوني Spiral development

في هذه المقاربة يجري تمثيل الإجرائية بشكل حلزوني بدلاً من النظر إليها كتتالٍ للأنشطة مع إمكان العودة إلى الوراء. تمثل كل دورة في الحلزون مرحلة من مراحل الإجرائية. لا توجد في هذه المقاربة مراحل مثبتة (كالتحليل أو التصميم)، وإنما يجري اختيار الدورات في الحلزون حسب الحاجة. وهنا يجري تقدير المخاطرة وكيفية التعامل معها ضمن الإجرائية.



قطاعات النموذج الحلزوني:

1- **تحديد الأهداف:** يجري تحديد أهداف واضحة ومعينة للمرحلة.

2- **تقدير المخاطرة وإضعافها:** يجري تقدير المخاطرات وتحدد الأنشطة التي ستخفض من أهم هذه المخاطرات.

3- **التطوير والتحقق من الصلاحية:** يتم اختيار نموذج التطوير من بين النماذج العمومية الثلاث التي رأيناها آنفاً

4- **التخطيط:** تتم مراجعة المشروع والتخطيط للمرحلة التالية من الحلزون.

3- أنشطة الإجرائية

سبق أن ذكرنا في الفقرة 3.5. أن الأنشطة التي تتألف منها الإجرائية هي:

توصيف البرمجيات

هي الإجرائية التي تسمح بتحديد الخدمات المطلوبة من النظام، والقيود التي تقيد تطويره وتشغيله. إن هذا النشاط يشكل ما يسمى **"هندسة المتطلبات" requirement engineering**. تتألف إجرائية هندسة المتطلبات من الخطوات التالية:

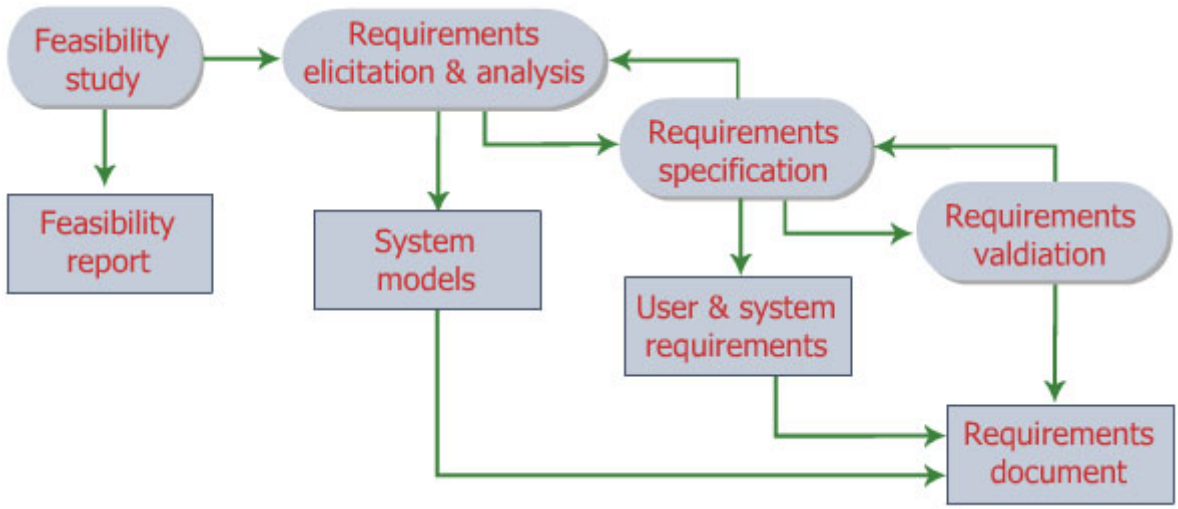
1- دراسة الجدوى.

2- تجميع المتطلبات وتحليلها.

3- توصيف المتطلبات.

4- التحقق من صلاحية المتطلبات.

يبين الشكل هذه الإجرائية، كما تظهر الوثائق الناتجة عن هذه الخطوات على الشكل (ضمن مستطيل).



تصميم البرمجيات وتنجيزها

في هذا النشاط نقوم بإجرائية تهدف إلى تحويل مواصفات النظام التي حددناها في المرحلة السابقة إلى نظام قابل للتشغيل. ونلاحظ هنا أننا نتحدث عن نشاطين هما التصميم والتنجيز. يهدف التصميم إلى تصميم بنية البرمجية التي تحقق المواصفات، في حين يهدف التنجيز إلى ترجمة هذه البنية إلى برنامج قابل للتشغيل وكثيراً ما تتداخل الأنشطة المتعلقة بالتصميم والتنجيز فيما بينها.

أنشطة إجرائية التصميم

تتألف إجرائية التصميم من الأنشطة التالية:

1 تصميم البنيان.

2- التوصيف الموجز.

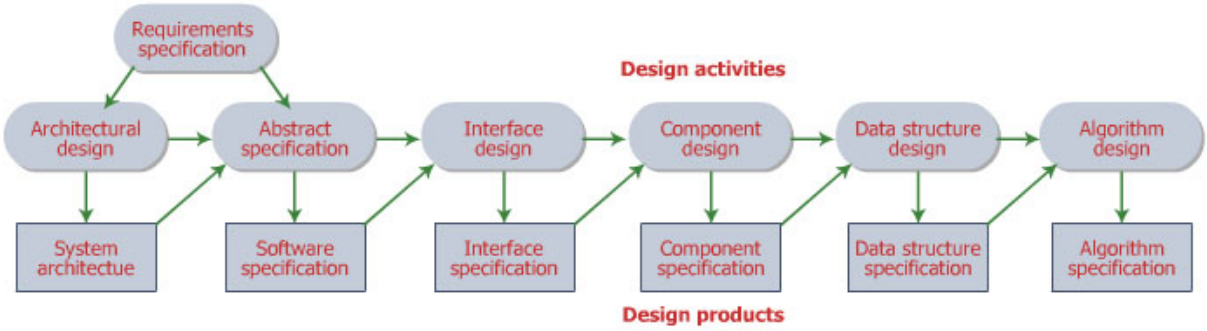
3- تصميم الواجهات.

4- تصميم المكونات.

5- تصميم بنى المعطيات.

6- تصميم الخوارزميات.

يبين الشكل هذه الإجرائية، كما تظهر الوثائق الناتجة عن هذه الخطوات على الشكل (ضمن مستطيل).



الطرائق البنيوية

هي مقاربات منهجية لتطوير وبناء تصميم للبرمجية. ويعبر عن التصميم عادة بمجموعة من النماذج البيانية. تتنوع النماذج بتنوع وجهة النظر والفلسفة المعتمدة في بنائها وفيما يلي بعض أنواع النماذج المستخدمة:

1- نموذج الأغراض.

2 نموذج التتالي.

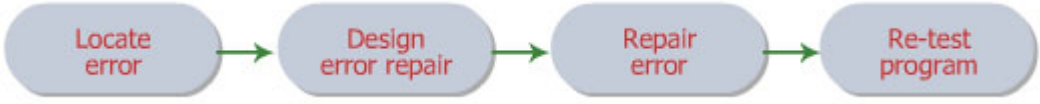
3- نموذج الانتقال بين الحالات.

4- النموذج البنيوي.

5- نموذج تدفق المعطيات.

البرمجة والتفلية

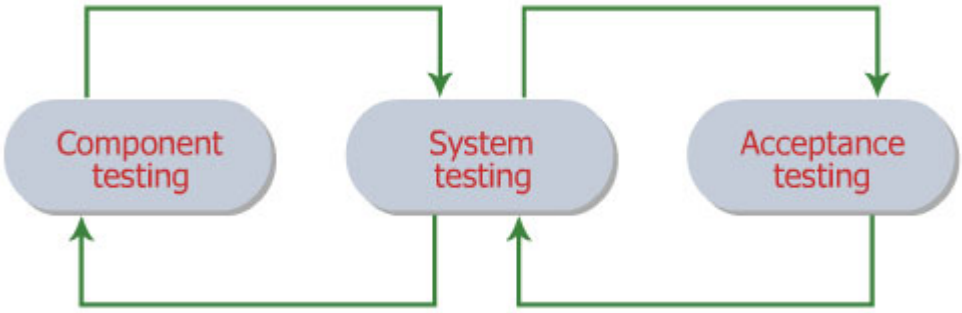
ذكرنا في بداية هذه الفقرة أن التنجيز يهدف إلى ترجمة التصميم إلى برنامج، وهنا علينا القيام بالبرمجة وبفحص البرنامج لإزالة جميع الأخطاء البرمجية التي قد يقع فيها المبرمج. إن عملية البرمجة نشاط شخصي إذ لا يوجد إجرائية عمومية للبرمجة. ويجري المبرمج عادة عدداً من الاختبارات للكشف عن الأخطاء التي يمكن أن يكون قد ارتكبها وإصلاحها وهذا ما يسمى إجرائية التفلية debugging. يبين الشكل إجرائية التفلية.



التحقق من صلاحية البرمجيات

تهدف عملية التحقق من الصلاحية validation والصحة verification (وهو ما يرمز له بـ V&V) إلى إثبات أن النظام يحقق المواصفات التي بني انطلاقاً منها وأنه يحقق متطلبات الزبون. وتتضمن هذه العملية إجرائيات تدقيق ومراجعة واختبارات للنظام.

يبين الشكل إجرائية الاختبار.



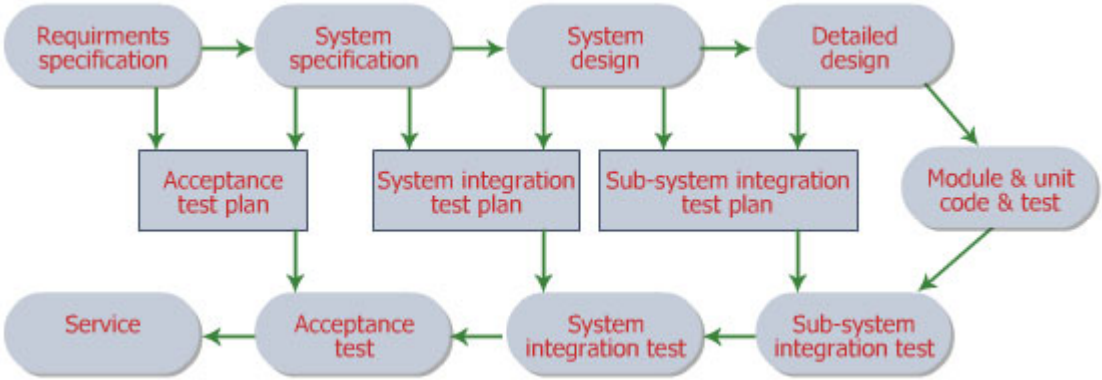
لاختبار النظام يجري إعداد عدد من حالات الاختبار المستنتجة من مواصفات المعطيات الحقيقية التي سيعالجها النظام لاحقاً، ثم يتم تنفيذ البرنامج باستخدام هذه الحالات.

تتألف إجرائية الاختبار من عدة خطوات:

1- اختبار المكونات (أو ما يسمى الاختبارات الأحادية): حيث تختبر المكونات كل على حده. ونقصد بالمكونات هنا التوابع أو الأغراض أو أية تجميع متماسك من هذه الكيانات.

2- اختبار النظام: حيث يختبر النظام ككل لاسيما خصائص الجودة كالأداء والجودة والأمن.

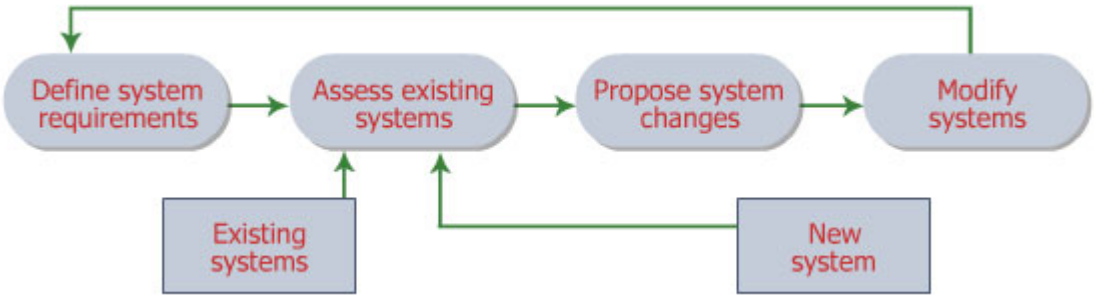
3- اختبارات القبول: حيث يختبر النظام باستخدام معطيات الزبون للتحقق من أن النظام يحقق احتياجات هذا الزبون. **يبين الشكل مراحل الاختبار**.



تطور البرمجيات

إن الطبيعة المطواعة للبرمجيات تجعل منها منتجات قابلة للتغيير. ونظراً للتغيرات المستمرة في عالم الأعمال فإن متطلبات الزبون تكون هي كذلك في تغير مستمر وبالتالي يجب أن ينتقل هذا التغيير إلى البرمجية التي تدعم الأعمال. وقد حاول العاملون في هذا المجال وضع حد فاصل بين التطوير (**البناء**) والتطور (**الصيانة**)، إلا أن هذا الأمر لم يحقق نجاحاً نظراً لأن معظم البرمجيات التي يجري العمل عليها هي بحد ذاتها برمجيات متطورة لأخرى سبقتها، لأن عدد البرمجيات الجديدة آخذ في التضاؤل.

يبين الشكل كيف يجري التطور على النظم البرمجية.

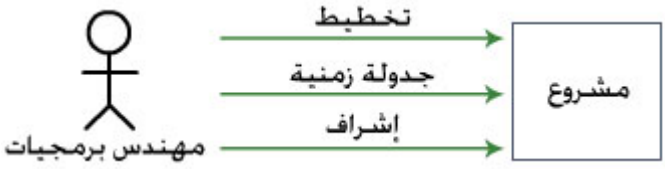


الفصل الثاني إدارة المشاريع البرمجية

1. إدارة المشاريع البرمجية

تُعتبر إدارة المشاريع البرمجية أحد المواضيع الهامة التي تعالجها هندسة البرمجيات، فسوء الإدارة يعني بالضرورة فشل المشروع بمعنى التأخر في تسليم البرمجية، وتجاوز الكلف المحددة قي البداية، وفشل البرمجية في القيام بالمتطلبات التي وضعت لأجلها.

يعتبر مدراء البرمجيات مسؤولين عن التخطيط لتطوير المشاريع وعن الجدولة الزمنية لها، كما أنهم يكونون مسؤولين عن الإشراف على العمل لضمان أنه يسير تبعاً للمعايير المطلوبة ويتقدم ضمن قيود الميزانية والمدة المحددتين للمشروع.



في الحقيقة يقوم مدراء المشاريع البرمجية بنفس العمل الذي يقوم به مدراء المشاريع الهندسية الأخرى، غير أن مهمتهم تختلف عن مهمة بقية المدراء في النقاط التالية:

1- إن المنتج الذي يشرفون على تطويره **غير محسوس intangible**.

ففي حين يمكن لمدراء المشاريع الأخرى رؤية مدى تطور العمل في المنتج بأعينهم (مدى ارتفاع البناء، أو مدى اكتمال صنع السفينة،...)، لا يستطيع مدراء المشاريع البرمجية أن يروا تطور العمل على البرمجية فهي غير مرئية وغير محسوسة، وإنما عليهم أن يعتمدوا على تقارير التطور التي يضعها العاملون.

2- إن الطبيعة المرنة والمطواعة للبرمجيات توحي بسهولة التعديل في أية لحظة ليتم استدراك أمر سبق نسيانه، غير أن هذا الأمر قد يقود إلى إجراء تعديلات كثيرة مما يجعل من الصعب مراقبة تطور سير العمل.

3- لا توجد معايير معترف بها عالمياً للإجرائيات البرمجية.

ففي فروع الهندسة الناضجة الأخرى، مكّنت الخبرة من وضع معايير جعلت إجرائية التطوير واضحة ومفهومة. أما إجرائيات تطوير البرمجيات، فهي تختلف من مؤسسة إلى أخرى. ورغم تحسن فهمنا لهذه الإجرائيات في السنوات الأخيرة، إلا أننا لا نزال غير قادرين على التنبؤ عما إذا كانت إجرائية تطوير ما قد تؤدي إلى مشاكل في التطوير. ويزداد هذا الأمر صعوبة عندما تكون هذه البرمجية جزءاً من منظومة أكبر.

4- إن كل مشروع برمجي ضخم يبدو فريداً من نوعه لا يتكرر.

وعليه، فمهما كانت خبرة الشركة القائمة على تطوير البرمجيات، فإنها قد لا تكون قادرة على التنبؤ بالمشاكل المحتملة، لاسيما مع التطورات السريعة في تقانة الحواسيب والاتصالات، حيث تصبح الخبرة القديمة عديمة النفع.

2. أنشطة إدارة المشروع

يصعب توصيف مهام مدير المشروع البرمجي، فهذا التوصيف معرض للتغيير والتبديل في بعض أجزائه حسب طبيعة المنتج الذي يجري تطويره والمؤسسة التي تقوم بالتطوير. غير أننا نورد هنا جميع المهام التي يمكن أن يقوم المدير ببعضها أو بها جميعاً في الأنشطة التالية:

1- كتابة مقترح المشروع.

وهي المهمة الأولى التي يقوم بها المدير عندما يقدم عرضاً بهدف الفوز بعقد لتطوير نظام برمجي.

2- التخطيط للمشروع ووضع جدول زمني له.

يحدد المدير الأنشطة التي يجب القيام بها ونقاط علّام المشروع والنواتج (راجع الفقرة 3.3 نقاط العلّام والنواتج) التي سيقدمها المشروع. كما يضع خطة لمتابعة تنفيذ المشروع.

3- تقدير كلفة المشروع: يقدر المدير الكلف والموارد اللازمة لتنفيذ خطة المشروع.

4- مراقبة العمل في المشروع وإجراء المراجعات.

وهي من المهام المستمرة في إدارة المشروع إذ يقوم المدير بمقارنة الخطة الموضوعة والكلف المتوقعة مع واقع تطور سير العمل والمبالغ المصروفة. كما يجري عدداً من المراجعات الرسمية لمراجعة تطور سير العمل في كامل المشروع والتأكد من أن المشروع الجاري العمل به يتفق مع المطلوب.

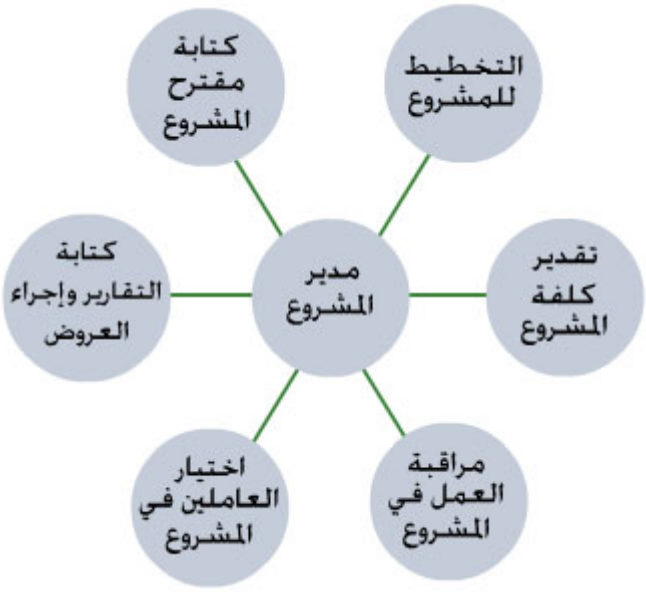
5- اختيار العاملين في المشروع وتقييمهم.

يقوم المدراء باختيار فريق العمل. ويجب أن يدرك المدير أنه قد لا يتمكن من توظيف الأشخاص المناسبين وذوي للخبرة للعمل في المشروع لأسباب عدة:

- لا تتحمل ميزانية المشروع الأجور العالية للخبراء.
- لا يتوفر فريق بالخبرة المطلوبة.
- ترغب شركة التطوير بتدريب فريقها الخاص وتكوين مهارات جديدة لديها من خلال المشروع.

كتابة التقارير وإجراء العروض والمحاضرات.

يعتبر المدير مسؤولاً عن إعداد تقارير وإجراء عروض تبين تقدم العمل للزبون.



3. التخطيط للمشروع

تقوم الإدارة الفعالة للمشروع البرمجي على التخطيط الجيد لتطور المشروع. فالمدير الناجح يجب أن يستبق الأمور ويتوقع الإشكالات الممكنة، ويحضر لها الحلول. وتعتبر عملية التخطيط من أكثر أنشطة إدارة المشروع استهلاكاً للوقت، وتبدأ منذ لحظة التفكير بالمشروع وتنتهي عند تسليم المشروع.

يضع المدير الخطة (راجع الفقرة 3.2 خطة المشروع) المتعلقة بالجدول الزمني والميزانية، كما يمكنه أن يستعين بأنواع أخرى من الخطط التي تدعم تلك الخطة (انظر الجدول). يبدأ المدير بوضع الخطة الفضلى التي يعتقد أنها ستؤدي إلى نجاح المشروع، ويجب أن تُراجع هذه الخطة مراجعة منتظمة عند حدوث أية مستجدات، ومع كل مراجعة يمكن أن تعدل الخطة إذا دعت الحاجة لذلك.

أنواع الخطط	الوصف
خطة الجودة quality plan	تصف إجراءات ومعايير الجودة التي سيتم استخدامها في المشروع.
خطة التحقق من الصلاحية validation plan	تصف المقاربة والموارد والخطة الزمنية اللازمة للتحقق من صلاحية النظام.
خطة إدارة التشكيلة configuration management plan	تصف إجراءات إدارة التشكيلة والبنى اللازمة لذلك.
خطة الصيانة maintenance plan	تتضمن توقعات لمتطلبات صيانة النظام وكلف الصيانة والجهود المطلوبة.
خطة تطوير الفريق staff development plan	تصف كيفية تطوير مهارات وخبرة أعضاء فريق عمل المشروع.

1- إجرائية تخطيط المشاريع

يمكننا التعبير عن إجرائية تخطيط المشاريع البرمجية بطريقة كتابة الخوارزميات البرمجية (شبه الرماز) على الشكل التالي:

1- ضع القيود الخاصة بالمشروع

2- قم بإجراء تقدير أولي لموسطات المشروع

3- حدد نقاط علام المشروع ونواتجه

4- **مادام** المشروع لم ينته أو لم يلغ **كرر**

4-1- ضع خطة زمنية للمشروع

4-2- أقلع الأنشطة وفق الخطة الزمنية

4-3- انتظر (لفترة)

4-4- أعد النظر في تقدم المشروع

4-5- صحح تقديراتك لموسطات المشروع

4-6- حدث الخطة الزمنية للمشروع

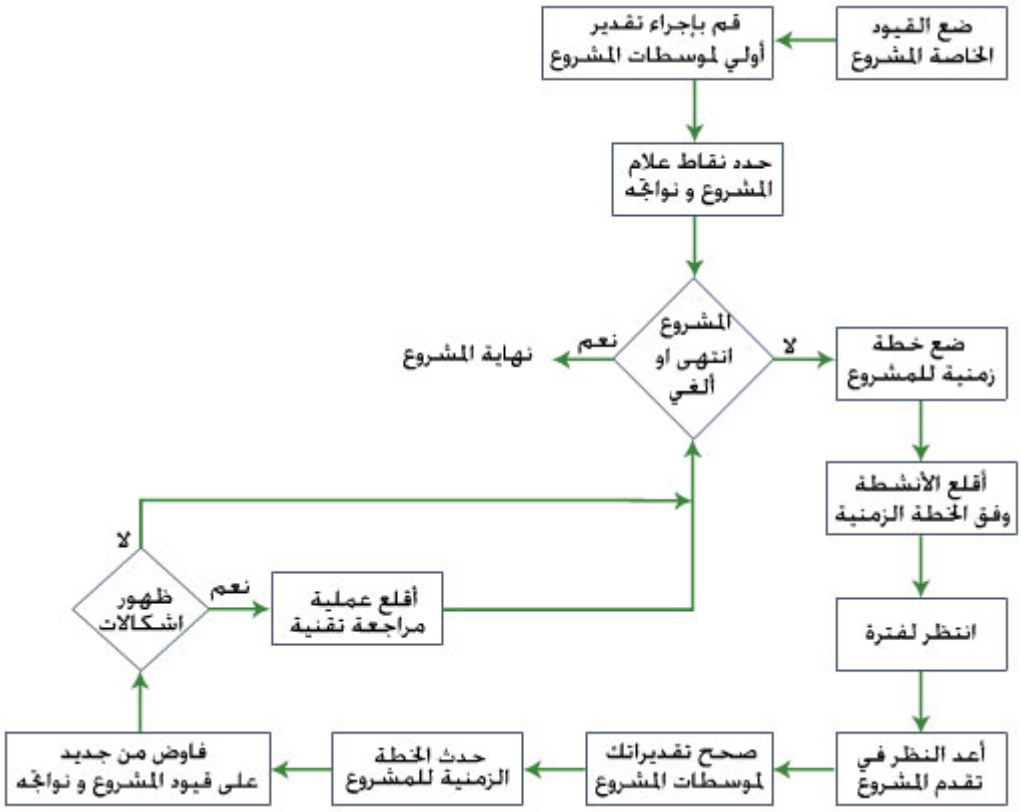
4-7- فاوض من جديد على قيود المشروع ونواتجه

4-8- **إذا** (ظهرت إشكالات)

4-8-1- أقلع عملية مراجعة تقنية

نهاية إذا

نهاية مادام



2- خطة المشروع

مع الشروع بالمشروع يبدأ المدير بإعداد خطته، ويركز فيها على الموارد المتوفرة للمشروع، والأجزاء التي يرى تجزئة العمل ضمنها، إضافة إلى الخطة الزمنية للعمل.

تتضمن خطة المشروع الفقرات التالية:

1- **مقدمة:** تصف أهداف المشروع والقيود المالية والزمنية التي تؤثر على إدارته.

2- **تنظيم المشروع**: تصف طريقة تنظيم فريق تطوير المشروع والأشخاص المعنيين ندور كل منهم.

3- **تحليل المخاطرة**: تصف المخاطرات التي قد يتعرض لها المشروع واحتمال وقوعها والإستراتيجية المعتمدة لتقليل المخاطرة.

4- **المتطلبات من الموارد المتعلقة بالتجهيزات والبرمجيات**: تصف التجهيزات اللازمة للتطوير مع البرمجيات الداعمة لها. وفي حال الحاجة لشرائها يجب تقدير كلفتها وتحديد الخطة الزمنية لاستلامها.

5- **تجزئة العمل**: تعرض كيفية تجزئة المشروع إلى أنشطة، وتحدد نقاط العلام والنواتج.

6- **الخطة الزمنية للمشروع**: وهي تبين العلاقة بين الأنشطة والزمن اللازم للوصول إلى كل نقطة علام وتكليف الأشخاص بالأنشطة.

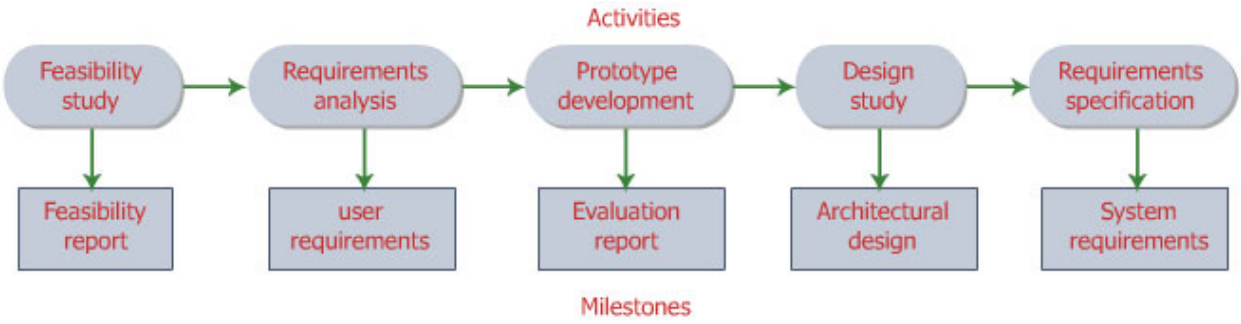
7- **آليات المراقبة ورفع التقارير**: تعرف التقارير التي يجب أن تنتج عن إدارة المشروع ومتى يجب أن تقدم والآلية المعتمدة لمراقبة المشروع.

3- نقاط العلام والنواتج

يجب أن تنظم الأنشطة في المشروع بحيث تنتج مخرجات محسوسة تسمح للإدارة بتقدير مدى تقدم العمل. من أجل تحقيق هذا الهدف يعرف مدير المشروع مجموعة من **نقاط العلام milestones** حيث تعتبر نقطة العلام بأنها نقطة النهاية في نشاط ما من أنشطة الإجرائية البرمجية. عند كل نقطة علام، يجب أن يكون هناك خرج رسمي (كتقرير مثلاً) يقدم للإدارة، وتكون هذه التقارير عادة موجزة تبين ما تم إنجازه. تمثّل نقطة العلام نهاية مرحلة محددة وواضحة في المشروع.

أما **المخرجات deliverables** فهي النواتج التي تنتج عن المشروع والتي يجري تسليمها للزبون. يكون التسليم عادة في نهاية مرحلة رئيسية (كالتوصيف أو التصميم). توافق المخرجات عادة نقطة علام، لكن العكس غير صحيح.

يقتضي تعريف نقاط العلام أن تجزأ الإجرائية البرمجية إلى أجزاء أساسية ترتبط بها مخرجات كما يظهر في الشكل التالي، حيث تظهر الأنشطة المعتمدة في هندسة المتطلبات.



4. وضع جدول زمني للمشروع

يعتبر وضع جدول زمني للمشروع من أصعب الأعمال التي يقوم بها مدير المشروع. ولكي يتمكن من القيام بها، عليه أن يبدأ بتقسيم المشروع إلى مهام، ثم يقدر الزمن اللازم لإنجاز كل واحدة من هذه المهام على حده، ثم يرتب هذه المهام بشكل **تسايري concurrent** ليحقق الاستفادة المثلى من القوة العاملة. ومن الضروري أن يسعى المدير إلى التقليل ما أمكن من تبعية المهام بعضها لبعض لأن هذه التبعية تتسبب بجعل بعض المهام تنتظر بعضها الآخر مما يؤدي إلى حدوث تأخير.

في الواقع يتأثر الجدول الزمني كثيراً بخبرة المدير وحدسه. يبين المخطط التالي إجرائية وضع الجدول الزمني للمشروع.

1- صعوبات تقدير الجدول الزمني

تواجه المدير عدد من الصعوبات أثناء تقديره للجدول الزمني. يمكن تلخيص هذه الصعوبات فيما يلي:

1- ليس من السهل أن يقدر المدير درجة صعوبة المسائل وكاف تطوير الحلول المطلوبة.

2- لا توجد علاقة تناسب بين عدد الأشخاص العاملين في مهمة وإنتاجيتهم.

3- إن إضافة أشخاص لمشروع متأخر يزيده تأخيراً بسبب زيادة حجم التواصل المطلوب.

4- تقع دوماً الأحداث غير المتوقعة. لذا، يجب أن يأخذ المدير بعين الاعتبار حالات الطوارئ ويسمح بها.

2- المخططات الشريطية وشبكات الأنشطة

هي مخططات بيانية تستخدم في التعبير عن الجدول الزمني للمشروع، وتُظهر المهام التي حددها مدير المشروع والتي تشكل بمجموعها أعمال المشروع. ينبغي الحرص على أن يكون حجم هذه المهام معقولاً (من رتبة أسبوع أو أسبوعين). تستخدم **شبكات الأنشطة activity networks** في إظهار العلاقات بين المهام وفي تحديد المسار الحرج، في حين تستخدم مخططات **الأعمدة bar charts** في إظهار الأنشطة بدلالة التاريخ.

لإيضاح هذين النوعين من المخططات، تأمل في المثال التالي:

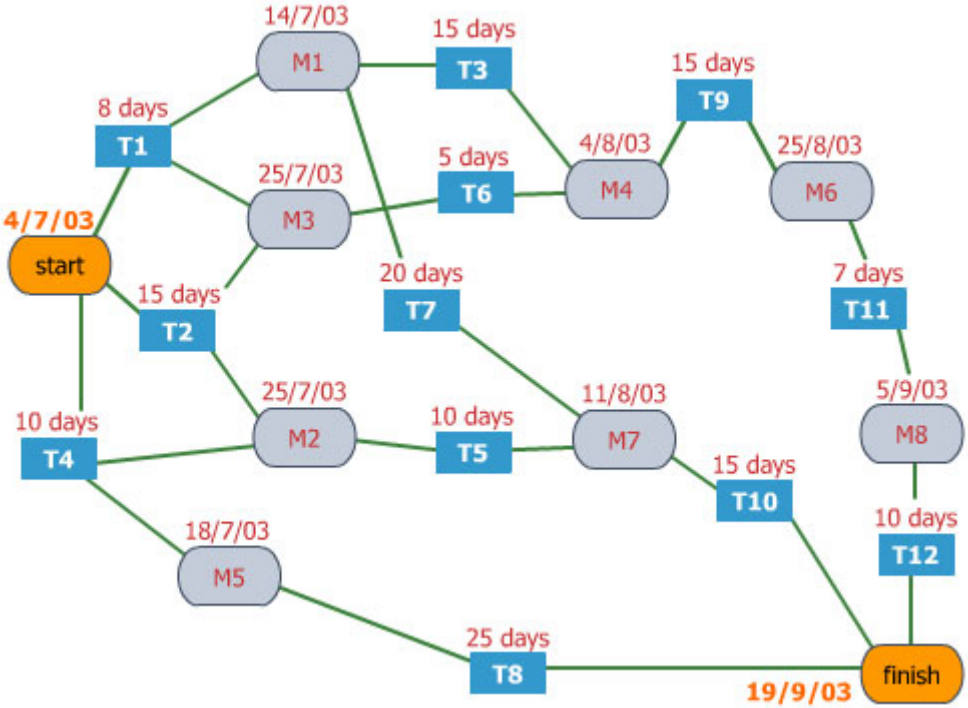
ليكن لدينا مشروعاً قام مديره بتجزئة أعماله ضمن المهام المبينة في الجدول التالي. يظهر الجدول رمز المهمة ومدتها والمهمة التي يجب أن تسبقها.

النشاط (المهمة)	المدة الزمنية(الأيام)	التبعيات
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

يبين الجدول السابق أن المهمة **T3** تتبع أو تلي المهمة **T1**. وهذا يعني أن **T1** يجب أن تنتهي قبل أن تبدأ **T3**. فمثلاً يمكن أن تكون **T1** هي مهمة التحضير لتصميم مكون ما، و**T3** هي تنجيز هذا التصميم، وبالطبع يجب أن ينتهي التصميم قبل بدء التنجيز.

انطلاقاً من هذا الجدول يمكن توليد شبكة الأنشطة كما في الشكل التالي. وهذه الشبكة تبين المهام التي يمكن أن تنفذ على التوازي مع غيرها، وتلك التي يمكن أن تنفذ على التسلسل بسبب تبعيتها لمهمة سابقة

لها. يرمز للأنشطة بمستطيلات أما نقاط العلام والنواتج فيرمز لها بأشكال بيضوية. أما التاريخ فهو يبين تاريخ بداية النشاط. إن الأداة المستخدمة في توليد هذه الشبكة من الجدول تحرص على ضرورة انتهاء كل نشاط بنقطة علام. ولا يبدأ النشاط قبل بلوغ نقطة العلام السابقة له وإنجازها.

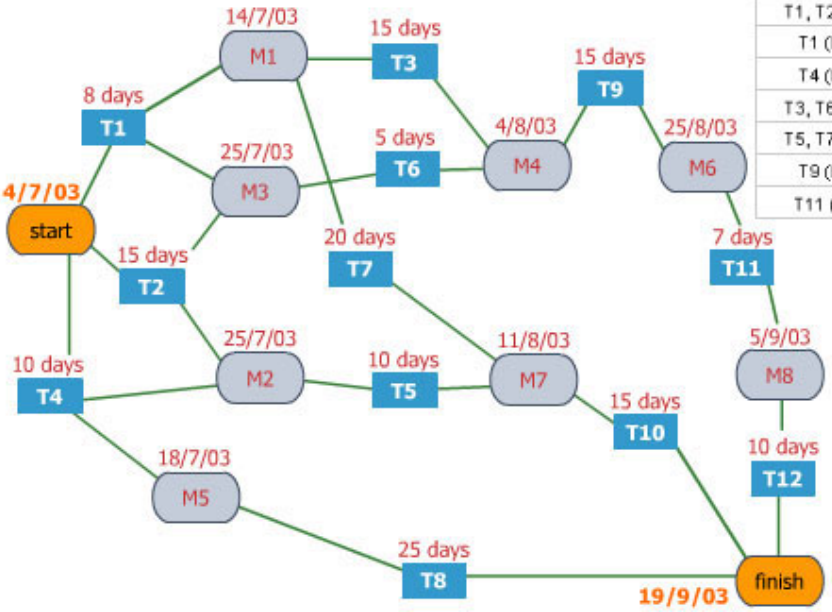


وقبل أن يبدأ الانطلاق من نقطة علام ما، يجب أن تكون جميع المسارات التي تؤدي إليها قد أنجزت .

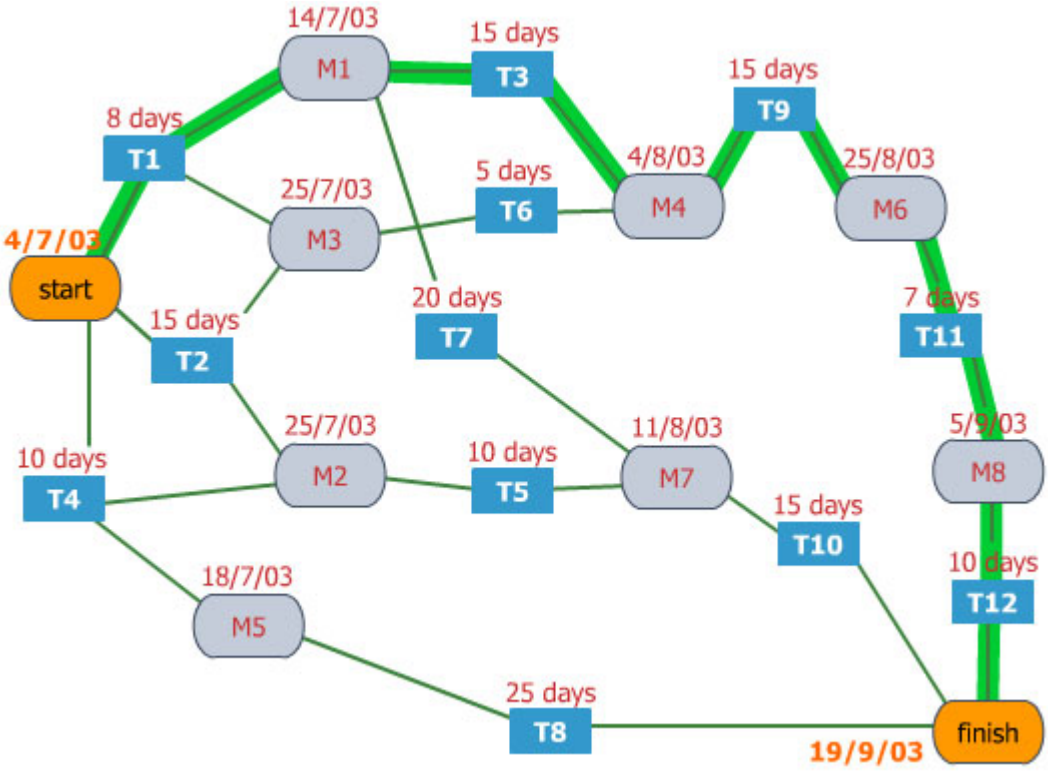
فمثلاً يجب أن تنجز المهام T3 و T6 قبل أن تبدأ T9.

تحويل الجدول إلى شبكة أنشطة:

النشاط (المهمة)	المدة الزمنية (الأيام)	التبعيات
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)



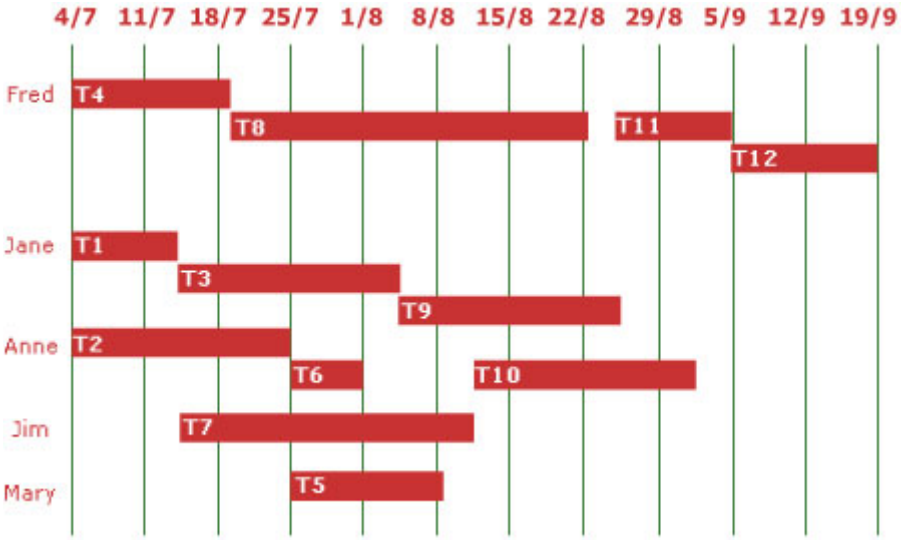
تحسب المدة الزمنية الصغرى التي يمكن أن ينتهي خلالها المشروع من أطول مسار في شبكة النشاطات، ويسمى هذا المسار **بالمسار الحرج** critical path. في مثالنا هذا يبلغ هذا المسار 11 أسبوعاً أو 55 يوم عمل، وهو المسار: Start-T1-M1-T3-M4-T9-M6-T11-M8-T12-Finish. يمثل هذا المسار الزمن اللازم لإنهاء المشروع، وأي تأخير في أحد الأنشطة الموجودة على مساره يؤدي حتماً إلى تأخير في تسليم كامل المشروع.



يبين الشكل التالي طريقة أخرى لعرض معلومات الجدول الزمني للمشروع. يسمى هذا النوع من المخططات المخطط **الشريطي** bar chart، فكل نشاط يبدأ على الشكل ويليه شريط يتناسب طوله مع المدة الزمنية اللازمة للمهمة.

3- إسناد المهام للأفراد

إضافة إلى مهمة وضع الجدول الزمني الخاص بالمشروع، يجب على المدير أن يهتم بإسناد المهام للأفراد العاملين في فريق العمل. تضاف هذه المعلومات إلى سابقاتها وتستطيع الأداة أن تولد مخططاً شريطياً يوضح وقت الحاجة لتوظيف أفراد في المشروع، فليس من الضروري أن يتم توظيف العاملين من بداية المشروع وحتى نهايته، بل يمكن أن يعمل بعض الأفراد في مشاريع أخرى عندما لا تكون هناك حاجة لهم في هذا المشروع، أو أن يستفيدوا من دورة تدريبية في هذا الوقت.



5. إدارة المخاطرة

أخذت إدارة المخاطرة مؤخراً حيزاً أكبر من اهتمام مدراء المشاريع. تعنى إدارة المخاطرة بتحديد المخاطرات ووضع الخطط للتقليل ما أمكن من أثر هذه المخاطرات على المشروع.

تعرف **المخاطرة** risk بأنها احتمال **حدوث أمر يعترض سير المشروع، ويمكنه أن يتسبب في تأخيره أو زيادة كلفته**.

تصنف المخاطرات في ثلاثة أصناف:

1- **مخاطرات المشروع:** وهي المخاطرات التي تؤثر في الجدول الزمني للمشروع أو في الموارد المخصصة له (كخسارة مبرمج خبير).

2- **مخاطرات المنـتج:** وهي المخاطرات التي تؤثر في جودة المنتج البرمجي أو في أدائه (كفشل مكون برمجي تم شراؤه في أداء ما هو متوقع منه).

3- **مخاطرات الأعمال:** وهي المخاطرات التي تؤثر في الشركة التي تطورالبرمجية أو توفرها (كظهور منافس للشركة يقدم تقنية حديثة).

يعرض الجدول التالي بعض الأمثلة عن مخاطرات يمكن أن تحدث أثناء تطوير البرمجيات:

المخاطرة	تأثيرها	وصفها
تبدلات في فريق العمل	على المشروع	أعضاء الفريق ذوو الخبرة سيتركون العمل في المشروع قبل إنجازه
تغيرات إدارية	على المشروع	توجد تغيرات في الإدارة التنظيمية ولها أولويات مختلفة عن سابقتها
عدم توفر التجهيزات	على المشروع	لن يتم تسليم التجهيزات التي تعتبر أساسية للعمل في المشروع في الموعد المحدد
تغير المتطلبات	على المشروع والمنتج	توجد تغييرات كثيرة في المتطلبات أكثر من المتوقع سابقاً
تأخير في التوصيف	على المشروع والمنتج	لن تتوفر التوصيفات المتعلقة بالواجهات الأساسية كما هو الاتفاق حسب الجدول الزمني
ضعف تقدير للحجم	على المشروع والمنتج	يوجد ضعف في تقدير حجم النظام
ضعف أداء أدوات CASE	المنتج	لا تعمل أدوات CASE التي تدعم المشروع وفق الأداء المتوقع لها
تغير في التقانة	الأعمال	التقانة التي تشكل القاعدة التي يبنى عليها النظام أصبحت

		متخلفة أمام تقانة جديدة أخرى
منافسة المنتجات الأخرى	الأعمال	تم إنتاج منتج منافس في السوق قبل إنجاز النظام

ومما يجدر ذكره أنه يمكن أن تتداخل هذه المخاطرات، لذا يجب على المدير أن يحسن إدارتها، وأن يتحسب لها ويتخذ الخطوات اللازمة لتلافيها. وقد يحتاج إلى أن يضع خطة للطوارئ يستخدمها في حال حصول هذه المخاطرات. تتألف إجرائية إدارة المخاطرات من المراحل التالية:

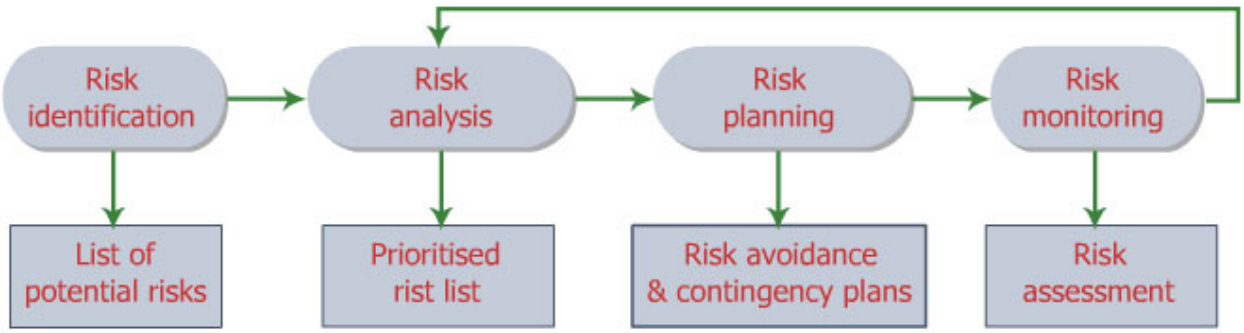
1- **تحديد المخاطرة:** يجري تحديد المخاطرات المحتملة المتعلقة بالمشروع والمنتج والأعمال.

2- **تحليل المخاطرة:** تقدر احتمالية وقوع هذه المخاطرات وعواقبها.

3- **التخطيط للمخاطرة:** توضع الخطط للتعامل مع المخاطرة سواء بتجنبها أو تقليل نتائجها على المشروع.

4- **مراقبة المخاطرة:** تقدر المخاطرة باستمرار وتراجع مخططات التخفيف من المخاطرة كلما توفرت معلومات حولها.

ويمكن أن نلخص إجرائية إدارة المخاطرات في الشكل التالي.



تعتبر إجرائية إدارة المخاطرة إجرائية تكرارية بمعنى أنها تستمر طوال فترة العمل في المشروع.

1- تحديد المخاطرة

وهي المرحلة الأولى من إدارة المخاطرة، وتُعنى باستكشاف المخاطرات المحتملة في المشروع. في هذه المرحلة لا تقيم المخاطر ولا تعطى أولويات. في الواقع يجري تعرف هذه المخاطر من خلال عمل جماعي يقوم به فريق عمل المشروع، وقد يستخدم الفريق مقاربة تقوم على "**العصف الذهني**" أو يتم تحديد المخاطرات اعتماداً على الخبرة. ويمكن أن تُعد لائحة تفقد لأنواع المخاطرات المحتملة. توجد على الأقل ستة أنواع من المخاطرات التي يمكن أن تظهر وهي:

1- **مخاطرات التقانة**: مخاطرات ناتجة عن التجهيزات والبرمجيات المستخدمة في تطوير النظام.

2- **مخاطرات في البشر**: مخاطرات مرتبطة بالعنصر البشري في فريق التطوير.

3- **مخاطرات تنظيمية**: مخاطرات تتعلق بالبيئة التنظيمية التي يجري تطوير البرمجية ضمنها.

4- **مخاطرات في الأدوات**: مخاطرات تتعلق بأدوات CASE أو أية برمجية داعمة مستخدمة في التطوير.

5- **مخاطرات في المتطلبات**: مخاطرات ناتجة عن تغييرات في متطلبات الزبون أو في إجرائية إدارة التغيير.

6- **مخاطرات في التقدير**: مخاطرات مرتبطة بتقديرات المدير لخصائص النظام وللموارد المسخرة لتطوير النظام.

يبين الجدول التالي أمثلة عن هذه المخاطرات:

نوع المخاطرة	المخاطرات المحتملة
في التقانة	1- لا تستطيع قاعدة المعطيات المستخدمة في النظام أن تعالج في الثانية الواحدة العدد المتوقع من المداولات. 2- تبين وجود خلل في المكونات البرمجية التي كان من المقرر إعادة استخدامها مما يحد من وظيفتها.
في البشر	1- لا يمكن توظيف فريق بالمهارات المطلوبة. 2- العناصر الرئيسية في الفريق مريضة وغير موجودة في الفترات الحرجة. 3- يحتاج فريق العمل إلى تدريب لا يمكن توفيره.
تنظيمية	1- جرت إعادة هيكلة المؤسسة بحيث تغيرت الإدارة المسؤولة عن المشروع. 2- تسببت المشاكل المالية التنظيمية بتخفيضات في ميزانية المشروع.
في الأدوات	1- إن الرماز الذي تولده أدوات CASE غير فعال. 2- لا يمكن تحقيق التكامل بين أدوات CASE.
في المتطلبات	1- تم اقتراح تغييرات على المتطلبات بحيث تحتاج إلى إعادة التصميم. 2- عجز الزبون عن فهم أثر التغييرات في المتطلبات.
في التقدير	1- يوجد ضعف في تقدير الزمن اللازم لتطوير البرمجيات. 2- يوجد ضعف في تقدير نسبة الأخطاء المصححة. 3- يوجد ضعف في تقدير حجم البرمجية.

2- تحليل المخاطرة

بعد وضع لائحة المخاطرات المحتملة، عليك أن تُقدر احتمال وقوع هذه المخاطرات، و مدى خطورتها على سير العمل في المشروع. يلجأ المدراء عادة إلى خبرتهم وحكمهم الشخصي في هذا التحليل.

يمكن أن يكون احتمال مخاطرة ما ضعيفاً جداً (<10%)أو ضـعيفاً(10-25%) أو متوسـطاً(25-50%) أو عالياً(50-75%) أو عالياً جداً(>75%)، كما يمكن أن يكون أثر المخاطرة كارثياً أو خطيراً أو يمكن تقبله أو مهملاً.

يبين الجدول التالي احتمال وقوع المخاطرات الواردة في الجدول السابق وأثرها على المشروع:

المخاطرة	الاحتمال	الأثر
تسببت المشاكل المالية التنظيمية بتخفيضات في ميزانية المشروع	ضعيف	كارثي
لا يمكن توظيف فريق بالمهارات المطلوبة للمشروع	عالي	كارثي
العناصر الرئيسية في الفريق مريضة وغير موجودة في الفترات الحرجة	متوسط	خطير
تبين وجود خلل في المكونات البرمجية التي كان من المقرر إعادة استخدامها مما يحد من وظيفتها	متوسط	خطير
تم اقتراح تغييرات على المتطلبات بحيث تحتاج إلى إعادة التصميم	متوسط	خطير
جرت إعادة هيكلة المؤسسة بحيث تغيرت الإدارة المسؤولة عن المشروع	عالي	خطير
لا تستطيع قاعدة المعطيات المستخدمة في النظام أن تعالج في الثانية الواحدة العدد المتوقع من المبادلات	متوسط	خطير
يوجد ضعف في تقدير الزمن اللازم لتطوير البرمجيات	عالي	خطير
لا يمكن تحقيق التكامل بين أدوات CASE	عالي	يمكن تقبله
عجز الزبون عن فهم أثر التغييرات في المتطلبات	متوسط	يمكن تقبله
يحتاج فريق العمل إلى تدريب لا يمكن توفيره	متوسط	يمكن تقبله
يوجد ضعف في تقدير نسبة الأخطاء المصححة	متوسط	يمكن تقبله
يوجد ضعف في تقدير حجم البرمجية	عالي	يمكن تقبله
إن الرماز الذي تولده أدوات CASE غير فعال	متوسط	مهمل

3- التخطيط للمخاطرة

تهتم إجرائية التخطيط للمخاطرة بوضع استراتيجية للتعامل مع كل مخاطرة من المخاطرات التي سبق تحديدها. وكما ذكرنا آنفاً يلجأ المدراء عادة إلى خبرتهم وحكمهم الشخصي في هذا التخطيط.

يمكن أن تندرج هذه الاستراتيجية تحت أحد العناوين التالية:

1- **استراتيجية التجنّب**: يهدف تطبيق هذه الاستراتيجية إلى تقليل احتمال حدوث هذه المخاطرة.

2- **استراتيجية التخفيف**: يهدف تطبيق هذه الاستراتيجية إلى التخفيف من أثر هذه المخاطرة.

3- **خطة الطوارئ**: يهدف تطبيق هذه الاستراتيجية إلى التعامل مع المخاطرة إن حدثت، ضمن خطة طوارئ.

يبين الجدول التالي أمثلة عن بعض الاستراتيجيات المعرفة للمخاطرات الواردة في جدول أنواع المخاطرة:

المخاطرة	الاستراتيجية
المشاكل المالية التنظيمية	حضر وثيقة موجزة للإدارة العليا تبين فيها كيف يشارك المشروع مشاركة فعالة في أهداف مجال الأعمال. (**استراتيجية الطوارئ**)
مشاكل التوظيف	أعِلم الزبون عن الصعوبات الكامنة واحتمال التأخير، وتحر إمكان شراء المكونات الجاهزة.
مرض أعضاء الفريق	أعِد تنظيم الفريق بحيث تكون هناك تقاطعات أكبر في العمل بين أعضاء الفريق وبذلك يفهم الأشخاص المهام الموكلة لزملائهم . (**استرتيجية التخفيف**)
مكونات ذات عيوب	استبدل المكونات ذات العيوب عبر شراء مكونات معروفة ذات موثوقية عالية. (**استراتيجية التجنب**)
تغييرات في المتطلبات	استنتج معلومات المتابعة بهدف تقدير أثر تغيير المتطلبات، وطبق مبدأ إخفاء المعلومات في التصميم قدر الإمكان.
إعادة الهيكلة التنظيمية	حضر وثيقة موجزة للإدارة العليا تبين فيها كيف يشارك المشروع مشاركة فعالة في أهداف مجال الأعمال.
فعالية قاعدة المعطيات	تحر إمكان شراء قاعدة معطيات ذات أداء أفضل.
ضعف في تقدير الزمن اللازم للتطوير	تحر إمكان شراء مكونات، وتحر إمكان استخدام مولّد برامج.

4- مراقبة المخاطرة

تعنى هذه الإجرائية بالتقدير المستمر للمخاطرات التي جرى تحديدها، لمعرفة التغيرات الحاصلة في إمكان حدوثها أو أثرها. بالطبع لابد من اللجوء إلى المؤشرات التي تعطينا المعلومات المطلوبة إذ لا يمكن القيام بهذا التقدير بالمراقبة المباشرة للعمل.

يظهر الجدول التالي بعض المؤشرات المستخدمة لمراقبة أنولع المخاطرات المحتملة.

نوع المخاطرة	المؤشرات الكامنة
في التقانة	التسليم المتأخر للتجهيزات والبرمجيات الداعمة، والعديد من المشاكل التقنية التي تم التعبير عنها.
في البشر	روح معنوية محبطة لدى فريق العمل، علاقات ضعيفة بين أعضاء فريق العمل، توفر شواغر.
تنظيمية	إشاعات تتعلق بالتنظيم، نقص القرارات لدى المدراء في السويات العليا.
في الأدوات	تردد الفريق في استخدام الأدوات، الشكوى من أدوات CASE ، الطلبات للحصول على محطات عمل ذات استطاعة أعلى.
في المتطلبات	العديد من طلبات تغيير المتطلبات، شكاوى الزبون.
في التقدير	الفشل في التوافق مع الجدول الزمني، الفشل في الحصول على تقارير واضحة عن أسباب الأعطال.

1. هندسة المتطلبات

يُطلق تعبير **هندسة المتطلبات على عملية تحديد الخدمات التي يطلبها الزبون من نظام ما والقيود التي سيطور ويعمل ضـمنها**. قد تكمن المشكلة الرئيسية التي نواجهها عند تطوير نظم برمجية ضخمة ومعقدة في هندسة المتطلبات.

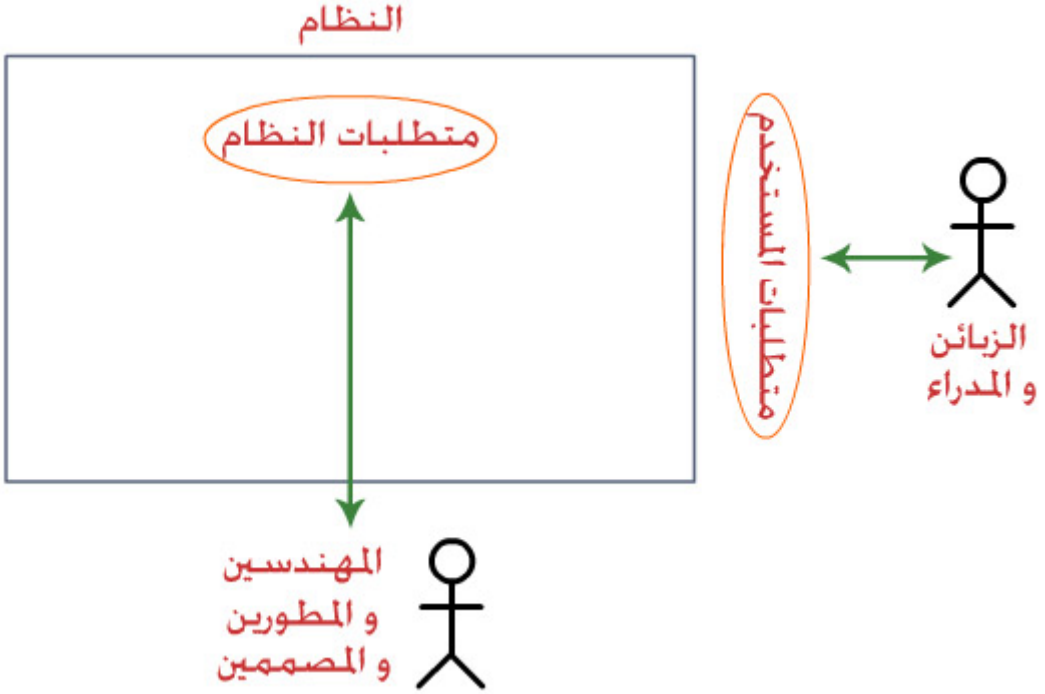
أما **المتطلبات فهي الوصف الكامل لخدمات النظام والقيود التي جرى تحديدها**.

يتراوح توصيف المتطلبات من (1) تعداد تجريدي على مستوى عالٍ للخدمات المطلوبة أو القيود المفروضة على نظام ما، إلى (2) توصيف تفصيلي رياضي وظيفي.

هذا التفاوت في التفصيل لا يمكن تجنبه لأن المتطلبات تؤدي دوراً مضاعفاً: فهي يمكن أن تكون أساساً لعرض مناقصة أو استدراج عروض فتكون مفتوحة للتفسير، أو أن تكون أساساً لعقد فيجب عندها أن تكون معرفة بالتفصيل.

يمكن أن تُكتب وثيقة المتطلبات بإحدى الطريقتين السابقتين. فإذا أرادت شركة ما أن تتعاقد على مشروع برمجي كبير فيجب أن تحدد حاجاتها بطريقة تجريدية كافية بحيث لا يكون الحل معرفاً سلفاً فيها. فتكتب المتطلبات بحيث يمكن أن يتقدم للعقد عدة عارضين قد يقدموا طرقاً مختلفة لتأمين حاجات الشركة. بعد أن ينجح أحد العارضين يجب أن يعرف النظام بالتفصيل الكامل بحيث يمكن للزبون فهمه والتأكد من عمله.

يمكن تقسيم المتطلبات إلى نوعين بحسب عموميتها ودرجة تفصيلها:



1- **متطلبات المستخدم**: وهي عبارة عن تعداد للخدمات وقيود العمل بلغة طبيعية مع مخططات توضيحية **موجهة أكثر للزبائن أو المدراء** الذين لا يهتمون بكيفية تنجيز النظام أو تفاصيل التسهيلات التي يوفرها.

مثال: يجب أن توفر البرمجيات طريقة لعرض الملفات الخارجية المنشأة بأدوات أخرى.

2- **متطلبات النظام**: وهي وثيقة بنيوية تعطي وصفاً مفصلاً لوظائف النظام وخدماته وقيود عمله تعرف ما يجب تنجيزه وتكون جزءاً من العقد بين الزبون والعارض. وهي موجهة **أكثر للمهندسين والمصممين والمطورين** الذين يحتاجون لمعرفة ما سيقوم به النظام بدقة. **أمثلة**:

- يجب أن يكون لدى المستخدم وسائل تسمح له بتعرف أنواع الملفات الخارجية.
- يمكن أن يكون لكل نوع من أنواع الملفات الخارجية أداة خاصة به.
- يمكن أن يُمثل كل نوع من أنواع الملفات الخارجية بأيقونة خاصة به يمكن للمستخدم تحديدها.
- عندما يختار المستخدم أيقونة تمثّل ملف خارجي يجب تطبيق الأداة الخاصة بنوع الملف على الملف.

كما يمكن تقسيم المتطلبات من حيث طبيعتها إلى متطلبات وظيفية ومتطلبات غير وظيفية.

2. المتطلبات الوظيفية والمتطلبات غير الوظيفية ومتطلبات نطاق العمل

1- المتطلبات الوظيفية

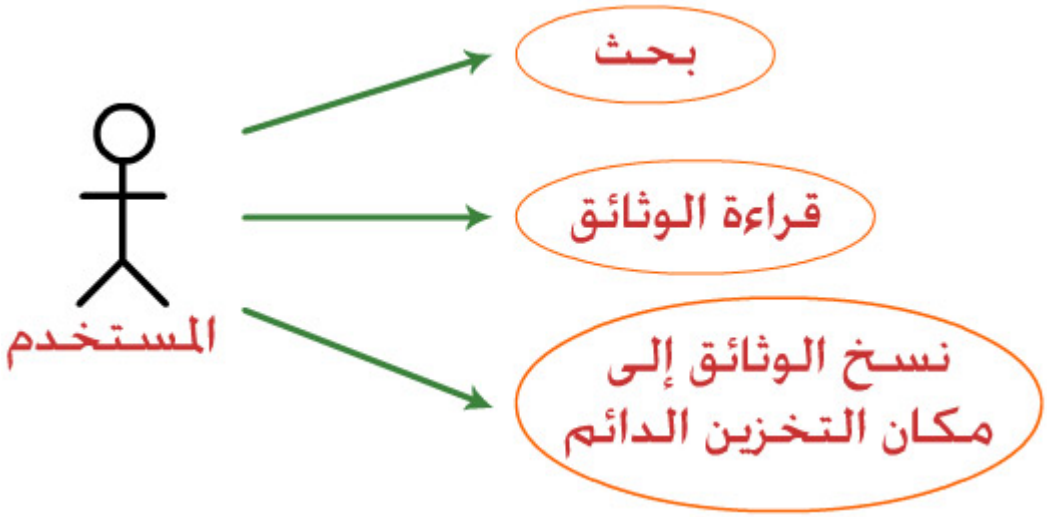
وهي الخدمات التي يجب أن يوفرها النظام وكيفية تفاعله مع مدخلات معينة وكيفية تصرفه في حالات خاصة. تختص المتطلبات الوظيفية بالخصائص التالية:

1- تصف وظائف وخدمات النظام.

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

3- يمكن أن تكون المتطلبات الوظيفية للمستخدم توصيفات عالية المستوى عن ما سيفعله النظام، لكن المتطلبات الوظيفية للنظام يجب أن تصف خدماته بالتفصيل.

سنستخدم كمثال للمتطلبات الوظيفية نظام مكتبة توفر واجهة لعدة قواعد معطيات من المقالات في مكتبات مختلفة. يمكن للمستخدم البحث عن مقالات معينة وتحميلها وطباعتها. في هذا المثال يمكن أن تكون لدينا المتطلبات الوظيفية التالية:



1- يجب أن يكون المستخدم قادراً على البحث في جميع قواعد المعطيات التي جرى تعريفها في البداية أو في مجموعة جزئية منها.

2- يجب أن يوفر النظام أدوات مناسبة لقراءة الوثائق الموجودة في مخزن الوثائق.

3- يجب أن يجري حجز رقم فريد لكل طلب بحيث يمكن للمستخدم نسخه إلى مكان التخزين الدائم الخاص به.

تظهر المشاكل عادةً عندما لا تكون المتطلبات محددة بدقة. فالمتطلبات غير الواضحة يمكن أن تُفسر بطرق مختلفة من قبل المستخدمين والمطورين.

على سبيل المثال، في **المتطلب (2)**، عبارة "**أدوات مناسبة**" قد يقصد المستخدم بها أداة قراءة خاصة لكل نوع من أنواع الوثائق بينما قد يفسرها المطور على أنها قارئ نصوص يُظهر محتويات الوثيقة. يجب أيضاً أن تكون المتطلبات كاملة تتضمن وصفاً لكافة التسهيلات المطلوبة، ومنسجمة لا تتضمن أية تناقضات أو تعارضات.

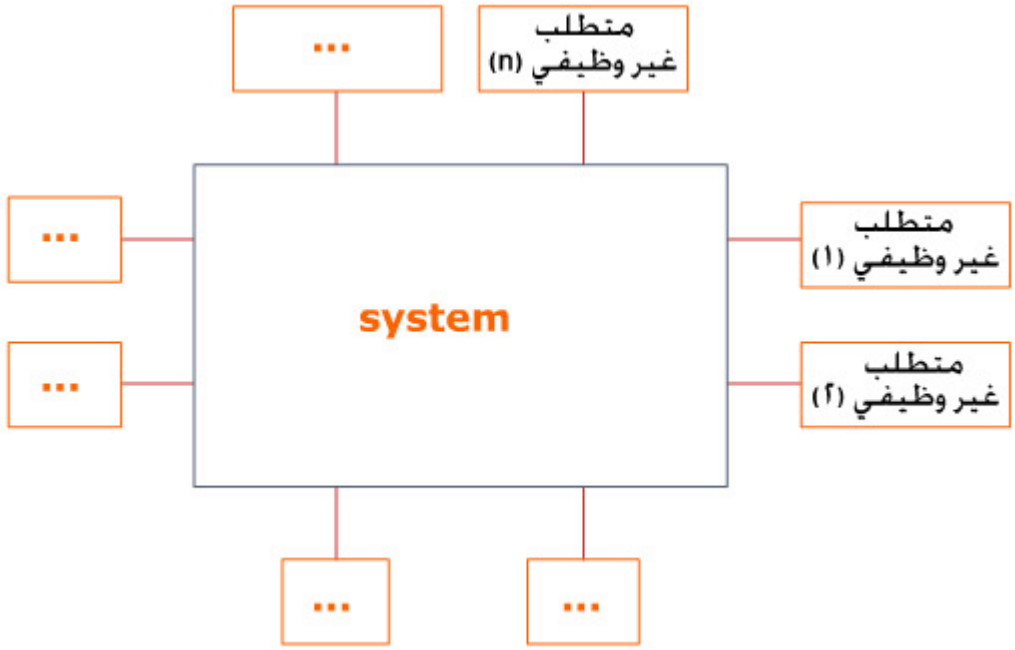
2- المتطلبات غير الوظيفية

وهي خصائص وقيود تتعلق بالوظائف والخدمات التي يوفرها النظام مثل القيود الزمنية والقيود على إجرائية التطوير والمعايير. فهي تتضمن مثلاً:

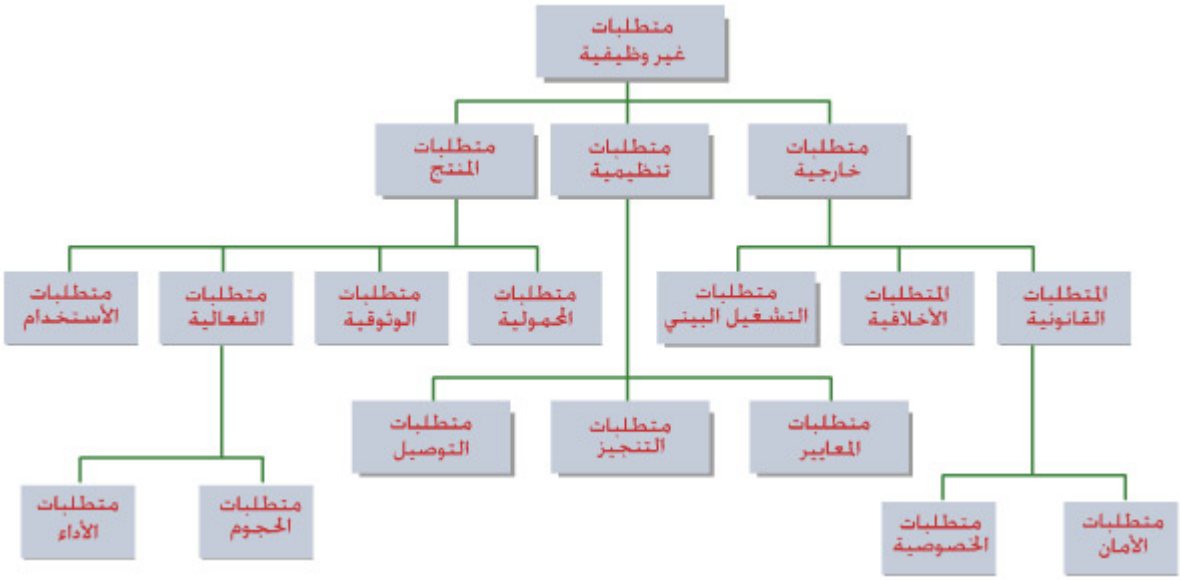
الوثوقية وزمن الاستجابة ومتطلبات التخزين والقيود على مقدرات تجهيزات الإدخال والإخراج والعرض.

يمكن أيضاً أن تتعلق بتحديد استخدام أداة معينة من الأدوات المساعدة في هندسة النظم CASE أثناء التطوير أو لغة برمجة أو طريقة تطوير محددة.

قد تكون المتطلبات غير الوظيفية أحياناً أهم من المتطلبات الوظيفية وقد يجعل عدمُ تحقيقها النظامَ عديم الفائدة.



يمكن تصنيف المتطلبات غير الوظيفية ضمن عدة أنواع (انظر الشكل):



1- **متطلبات المنتج**: تحدد خصائص المنتج النهائي كسرعة التنفيذ والوثوقية. **مثال**: يجب أن تُنجز واجهة الاستخدام في نظام المكتبة بلغة HTML بسيطة دون إطارات أو بريمجات جافا.

2- **متطلبات تنظيمية**: تنتج عن سياسات تنظيمية أو إجراءات كالمعايير الإجرائية المستخدمة أو متطلبات التنجيز. **مثال**: يجب أن تتوافق اجرائيات التطوير والوثائق الناتجة مع تلك المعرفة في المعيار **XYZCo-SP-STAN-95**.

3- **متطلبات خارجية**: تنتج عن عوامل خارج النظام وإجرائية تطويره كمتطلبات قابلية التشغيل البيني interoperability والمتطلبات القانونية. **مثال**: يجب أن لا يكشف النظام للمشغّلين عن أية معلومات شخصية تخص الزبائن زيادةً عن أسمائهم وأرقامهم المرجعية.

تختلف المتطلبات غير الوظيفية عن الأهداف. **فالأهداف هي مقاصد عامة للمستخدم** كسهولة الاستخدام تساعد المطور على اتخاذ بعض القرارات، أما **المتطلبات غير الوظيفية فيجب أن تكون قابلة للقياس والتأكد والاختبار**.

مثال

أحد أهداف النظام: يجب أن يكون النظام سهل الاستخدام من قبل الخبراء في العمل ويجب أن يكون مصمماً بطريقة تقلل من أخطاء المستخدمين.

المتطلب غير الوظيفي المقابل: يجب أن يكون الخبراء في العمل قادرين على استخدام كافة وثائق النظام بعد ساعتين من التدريب. ويجب أن لا يتجاوز عدد الأخطاء (2) وسطياً في اليوم.

يوضح الجدول التالي قياسات لبعض المتطلبات غير الوظيفية.

المتطلب	القياس
السرعة	1- عدد المناقلات المعالجة في الثانية 2- زمن الاستجابة لطلب المستخدم 3- زمن تحديث الشاشة
الحجم	1- ميغا بايت 2- عدد رقاقات ROM
سهولة الاستخدام	1- زمن التدريب 2- عدد إطارات المساعدة
الوثوقية	1- الزمن الوسطي لحدوث الأعطال 2- احتمال عدم إتاحة العمل 3- نسبة حدوث الأعطال
المتانة	1- الزمن اللازم لإعادة التشغيل بعد العطل 2- نسبة الأحداث المؤدية للأعطال 3- احتمال تخريب المعطيات عند العطل
المحمولية	1- نسبة التعليمات المتعلقة بالنظام 2- عدد الأنظمة المشمولة

من الطبيعي في الأنظمة المعقدة أن تتضارب بعض المتطلبات الوظيفية. عند حدوث ذلك يجب أن تكون هنالك أفضلية للمتطلبات الأهم والأكثر حرجاً.

3- متطلبات نطاق العمل

وهي التي تأتي من نطاق التطبيق وتعكس ميزاته في النظام. وهي يمكن أن تعبر عن متطلبات وظيفية جديدة أو قيود على متطلبات موجودة أو تعاريف لطرق حساب محددة. يمكن أن يكون النظام غير قابل للعمل إن لم يحقق متطلبات نطاق العمل.

أمثلة

1- يجب توفر واجهة استخدام معيارية تعتمد على المعيار Z39.50.

2- بسبب حقوق التأليف يجب حذف بعض الوثائق مباشرةً عند ورودها. اعتماداً على طلب المستخدم يمكن أن تُطبع هذه الوثائق على طابعة محلية أو شبكية.

تكون متطلبات نطاق العمل عادةً موصفة بلغة التطبيق التي اعتاد عليها المختصون في مجال العمل ولا تكون سهلة الفهم بالنسبة للمهندسين المطورين للنظام. **كما أن فهم المختصين لمجال عملهم واعتيادهم عليه قد يجعلهم لا يفكرون في وضع هذه المتطلبات بصراحة في وثيقة المتطلبات.**

3. متطلبات المستخدم

يجب أن توصف المتطلبات الوظيفية وغير الوظيفية بطريقة مفهومة من قبل مستخدمي النظام الذين لا يملكون معارف تقنية تفصيلية. تُعرف متطلبات المستخدم بلغة طبيعية مع مخططات توضيحية وجداول.

إن كتابة المتطلبات بلغة طبيعية تجعل منها أحياناً قليلة الوضوح وغير دقيقة، فمن الصعب كتابة متطلبات دقيقة مع المحافظة على سهولة قراءة النص. يمكن أيضاً أن يجري الخلط بين المتطلبات الوظيفية وغير الوظيفية أو تجميع عدة متطلبات مع بعضها مما يكون سبباً للالتباس، أو جمع متطلبات عامة مع تفاصيل دقيقة.

مثال

"**سوف يوفر نظام المكتبة نظام محاسبة مالية يحتفظ بسجلات الدفعات التي دفعها المستخدمون. يمكن لمدير النظام تغيير الإعدادات بحيث يحصل المستخدمون الدائمون على نسبة حسم**".

نجد في هذا المتطلب دمجاً لمفهوم نظام المحاسبة مع معلومات تفصيلية عن إعداده من قبل المدراء غير مناسبة على هذا المستوى.

توجيهات كتابة المتطلبات

1- اعتمد صيغة معيارية لجميع المتطلبات.

2- استخدم اللغة الطبيعية بطريقة متجانسة. استخدم مثلاً كلمة "**سيكون**" للمتطلبات الإجبارية وكلمة "**يجب**" للمتطلبات المرغوبة.

3- استخدم وسائل إظهار النص **كالكتابة بلون غامق** لتحديد الأجزاء الهامة من المتطلبات.

4- تجنب استخدام الاختصارات.

4. متطلبات النظام

توصيفات لوظائف النظام وخدماته وقيوده أكثر تفصيلاً من متطلبات المستخدم. ستكون هذه التوصيفات أساساً لتصميم النظام ويمكن أن تُضمن في العقد الخاص بالنظام، وتمثل باستخدام نماذج بيانية خاصة سنراها في فقرة لاحقة.

مبدئياً، يجب أن تحدد المتطلبات **ما** يجب أن يفعله النظام بينما يصف التصميم **كيف** يفعل ذلك. لكن عملياً لا يمكن فصل المتطلبات عن التصميم لأن بنية النظام قد تُستخدم لتفصيل المتطلبات أو قد يكون النظام سيعمل مع أنظمة أخرى مما يقيد تصميمه ويكون سبباً في فرض متطلبات خاصة عليه. كما أن استخدام تصميم معين للنظام قد يتعلق ببنيته أو أمنه أو متطلباته غير الوظيفية قد يكون أحد متطلبات النظام.

كما في متطلبات المستخدم يمكن أن تُكتب متطلبات النظام بلغة طبيعية لكن ذلك يجعلها صعبة الفهم وموضع التباس فتُفسر بأكثر من طريقة. كما أن مرونة اللغة الطبيعية تجعل من الممكن التعبير عن متطلب ما بأكثر من طريقة. وأخيراً فإن البنى المستخدمة في اللغة الطبيعية غير مناسبة لصياغة بنيوية للمتطلبات.

يمكن التفكير بعدة بدائل عن توصيف المتطلبات بلغة طبيعية مثل:

1- **لغة بنيوية**: تعتمد على نماذج وقوالب معيارية لتوصيف المتطلبات.

2- **لغة لوصف التصميم**: تشبه لغة البرمجة لكن بتجريد أعلى لتحديد المتطلبات بتعريف نموذج تشغيلي للنظام.

3- **تدوين بياني**: مخططات بيانية مدعمة بحواشي نصية تستخدم لتعريف المتطلبات الوظيفية مثل مخططات حالات الاستخدام use case diagrams ومخططات التسلسل sequence diagrams.

4- **توصيف رياضي**: يعتمد على مفاهيم رياضية كالآلات ذات الحالات المحدودة العدد، أو المجموعات التي تحدد بدقة وظائف النظام لكن معظم الزبائن لا يفهمونها.

1- كتابة المتطلبات بلغة بنيوية

إن وجود قوالب محددة لكتابة المتطلبات يحد من حرية كتابتها ويساهم في توحيد تعريفها بطريقة معيارية. كما يمكن الحد من المصطلحات المستخدمة في التوصيف. هذا الخيار مفيد جداً، فمع استمرار الاستفادة من القوة التعبيرية للغة يجري فرض درجة من النظامية على التوصيف كما في الشكل التالي.

برنامج التحكم بضخ الأنسولين	
الوظيفة	حساب جرعة الأنسولين: مستوى آمن للسكر.
الوصف	حساب جرعة الأنسولين التي يجب إعطاؤها عندما يكون مستوى السكر في الحدود الآمنة بين 3 - 7 وحدات.
المدخلات	قراءة السكر الحالية (**r2**) وقراءتي السكر السابقتين (**r0 and r1**).
المصدر	قراءة السكر الحالية من المُحِس. القراءات الأخرى من الذاكرة.
المخرجات	جرعة الأنسولين التي يجب إعطاؤها **CompDose**.
الوجهة	حلقة التحكم الرئيسية.
الفعل	تكون قيمة جرعة الأنسولين **CompDose** صفراً إذا كان مستوى السكر مستقراً أو متناقصاً أو تناقصت نسبة زيادته. إذا كان مستوى السكر يزداد وتزداد نسبة زيادته فيجري حساب **CompDose** بتقسيم الفرق بين مستوى السكر الحالي والمستوى السابق على 4 وتقريب النتيجة. إذا كانت النتيجة بعد التقريب صفراً تكون قيمة **CompDose** هي أصغر جرعة يمكن إعطاؤها.
المطلوب	القراءتين السابقتين بحيث يمكن حساب نسبة التغيير في مستوى السكر.
الشروط السابقة	يحتوي مستودع الأنسولين كمية تكفي للجرعة العظمى المسوح بها.
الشروط التالية	يجري استبدال **r0** بـ **r1** ثم **r1** بـ **r2**.
التأثيرات الجانبية	لا يوجد.

يمكن وضع استمارات وقوالب معيارية لتوصيف المتطلبات بعدة أشكال لكن يجب أن تتضمن هذه الاستمارات المعلومات التالية:

1- تعريف الوظيفة أو الكيان.
2- تعريف المدخلات وتحديد مصدرها (من أين تأتي).
3- تعريف المخرجات وتحديد وجهتها (إلى أين تذهب).
4- تحديد الكيانات المطلوبة أو المستخدمة.
5- وصف الفعل المطلوب عمله.
6- الشروط السابقة والشروط التالية (إذا كان ذلك مناسباً).

7- الآثار الجانبية للوظيفة (في حال وجودها).

التوصيفات الجدولية هي أيضاً إحدى الأدوات التي تدعم اللغة الطبيعية في توصيف المتطلبات خاصةً عند وجود عدة بدائل من سيرورات العمل انظر الجدول التالي.

الشرط	الفعل
مستوى السكر يتناقص (r2 < r1)	CompDose = 0
مستوى السكر مستقر (r2 = r1)	CompDose = 0
مستوى السكر يتزايد لكن نسبة تزايده تتناقص ((r2-r1)<(r1-r0))	CompDose = 0
مستوى السكر يتزايد لكن نسبة تزايده مستقرة أو متزايدة ((r2-r1) ≥ (r1-r0))	CompDose = round ((r2-r1)/4) If rounded result = 0 then CompDose = MinimumDose

2- النماذج البيانية

النماذج البيانية هي الأكثر جدوى عند ضرورة إظهار متطلبات تتعلق بتغييرات الحالة أو بسلسلة من الأعمال أو بكيفية إجراء الحسابات وتفاعل المستخدم مع النظام.

ترتبط هندسة المتطلبات ارتباطاً وثيقاً بعملية النمذجة حيث يقوم المحللون ببناء نماذج بيانية للنظام تساعد في فهم وظائفه والتواصل بشأنها مع الزبون.

تساعد الأنواع المختلفة من النماذج على وصف النظام من وجهات نظر مختلفة:

1- وجهة نظر خارجية تظهر السياق أو البيئة التي سيوجد فيها النظام.

2- وجهة نظر سلوكية تظهر سلوك النظام.

3- وجهة نظر بنيوية تظهر بنية النظام والمعطيات.

ينتج عن وجهات النظر السابقة عدة أنواع من النماذج مثل:

1- نماذج تقسيم النظام إلى أنظمة جزئية.

2- نماذج تفاعل النظام مع الأحداث.

3- نماذج معالجة المعطيات.

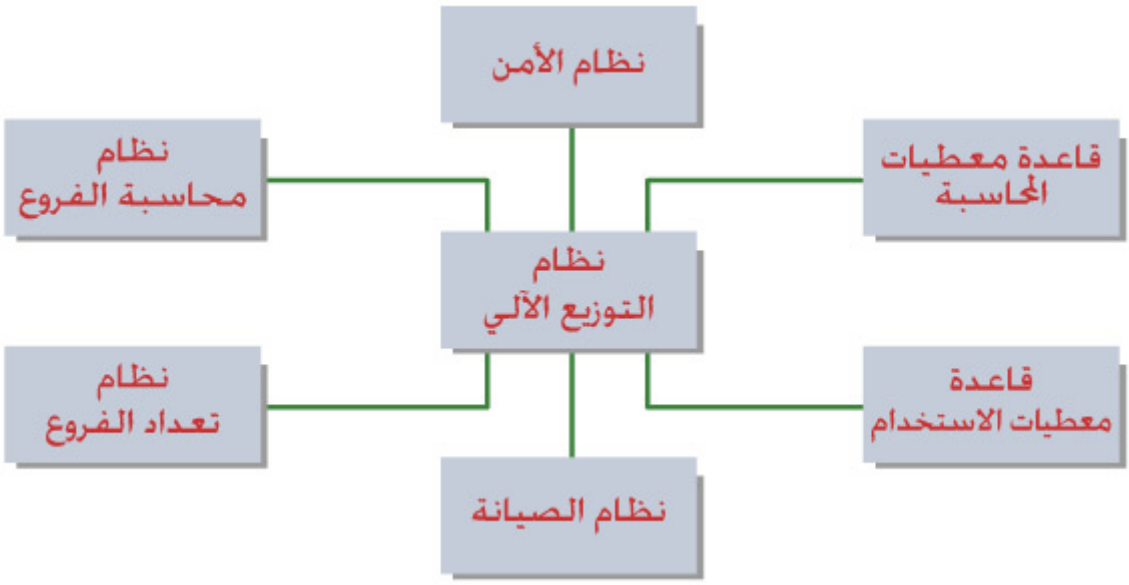
4- نماذج صفوف الكيانات وارتباطاتها.

نماذج السياق Context Models

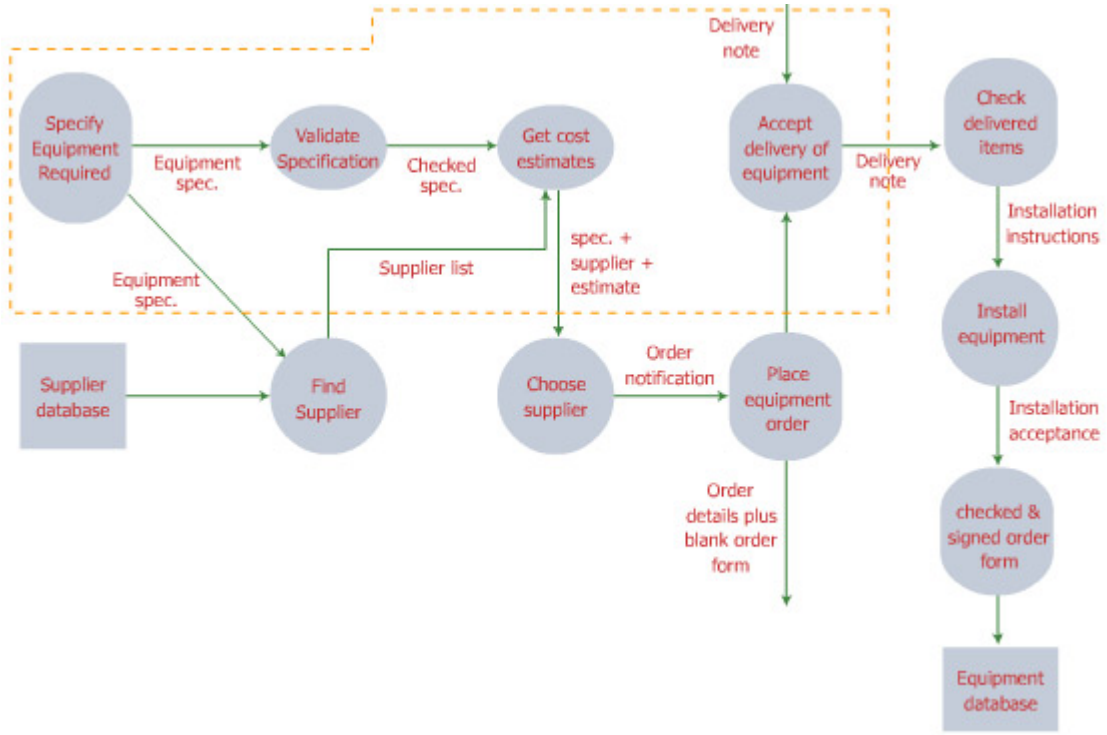
تُستخدم نماذج السياق لإظهار السياق التشغيلي للنظام والأشياء التي تقع خارج حدوده.

يمكن جداً لمؤثرات تنظيمية أو اجتماعية أن تساهم في تقرير حدود النظام. الخطوة الأولى بعد تعريف حدود النظام هي تعريف ارتباطاته مع البيئة والأنظمة الأخرى المحيطة به باستخدام **مخطط بنيان**

architecture diagram. يُظهر الشكل التالي نموذج بنيان لبنية نظام معلومات للتوزيع الآلي يتضمن شبكة آلات توزيع مصرفية أوتوماتيكية ATM.



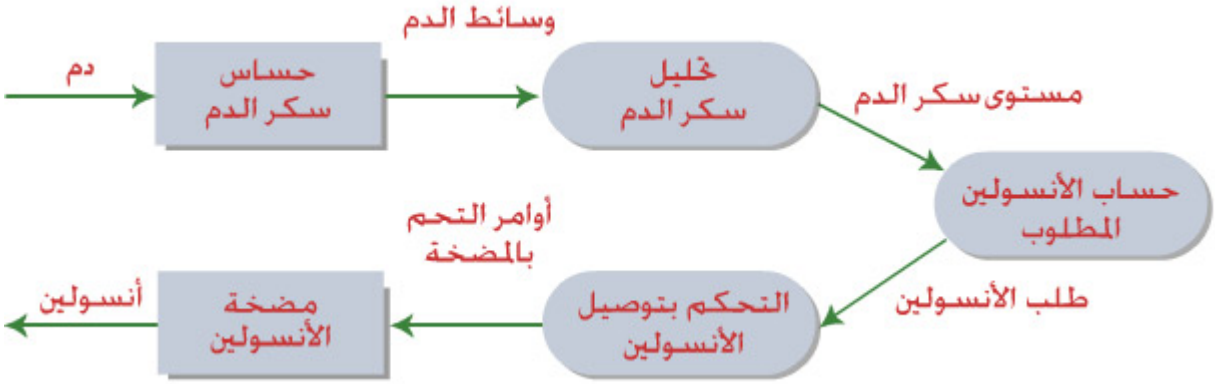
يمكن أيضاً التعبير عن حدود النظام **بنماذج إجرائية Process Models** تُظهر الإجرائيات الموجودة في النظام مع توضيح حدوده وسياقه (الخط المنقط في الشكل التالي: نموذج إجرائي لعملية الحصول على التجهيزات).



النماذج السلوكية

وهي النماذج التي تستخدم لوصف سلوك النظام إما عن طريق (1) نماذج معالجة المعطيات أو (2) نماذج حالة الآلة التي تظهر استجابة النظام للأحداث.

يمكن تمثيل نماذج معالجة المعطيات **بمخططات تدفق المعطيات** Data Flow Diagrams DFD وهي مخططات بسيطة وسهلة الفهم على الزبائن وتظهر المعالجة الكاملة للمعطيات من الناحية الوظيفية، كما يمكن استخدامها لتمثيل تبادل المعطيات مع أنظمة أخرى انظر الشكل التالي (مخطط تدفق المعطيات الخاص بضخ الأنسولين).

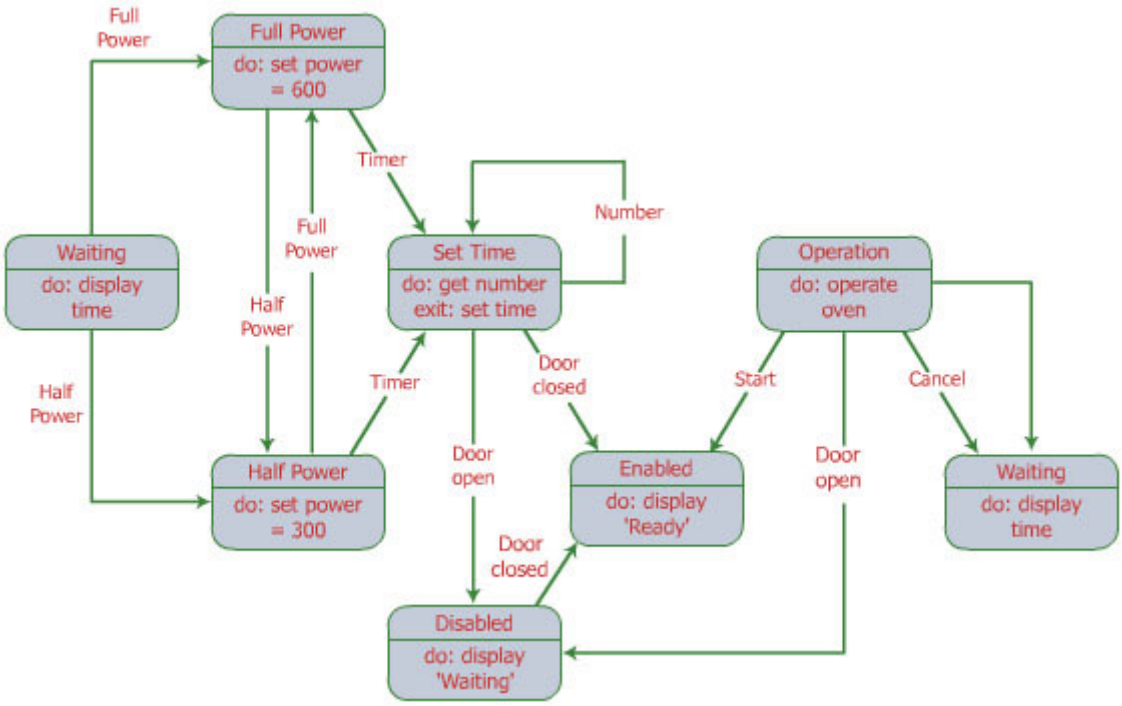


النماذج السلوكية

أما نماذج حالة الآلة التي تُظهر استجابة النظام للأحداث الداخلية والخارجية فيمكن تمثيلها **بمخطط ـات الحالة** State charts التي تُعتبر جزءاً من لغة UML. تُستخدم هذه المخططات غالباً لتمثيل نظم الزمن الحقيقي. حيث يجري تمثيل حالات النظام بعقد والأحداث بوصلات وعند حدوث حدث ينتقل النظام من حالة إلى أخرى انظر الشكل التالي (مخطط الحالة الخاص بفرن أمواج ميكروية بسيط). كما تسمح بإظهار تقسيمات النظام إلى نظم جزئية. يمكن وضع وصف مختصر لكل حالة واستكمال المعلومات بجداول تصف الحالات والأحداث.

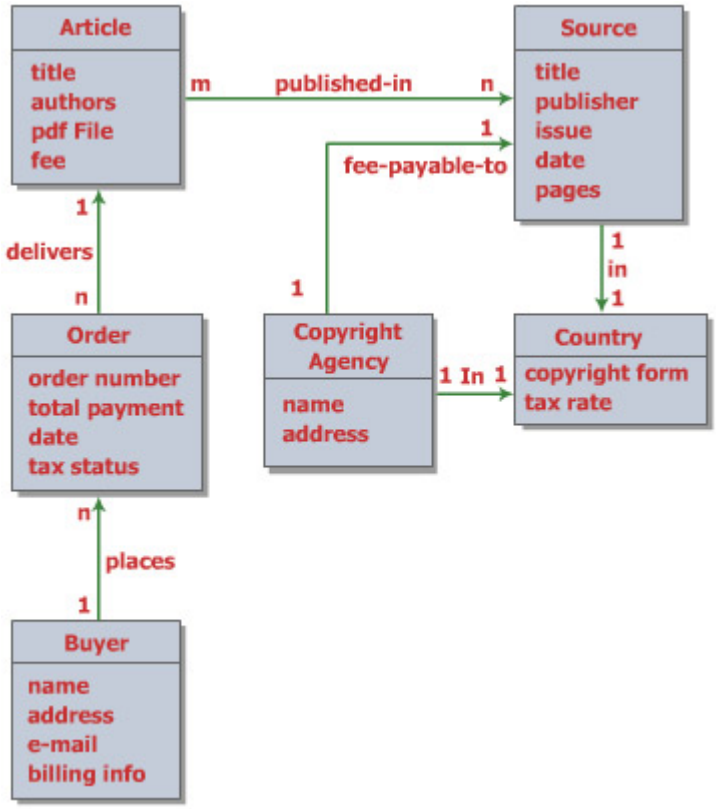
الحالة	الوصف
انتظار	الفرن بانتظار المدخل. تُظهر الشاشة الوقت الحالي.
نصف طاقة	طاقة الفرن 300 واط. تُظهر الشاشة "نصف طاقة".
طاقة كاملة	طاقة الفرن 600 واط. تُظهر الشاشة "طاقة كاملة".
تحديد الوقت	يأخذ وقت الطبخ القيمة التي يُدخلها المستخدم. تُظهر الشاشة الوقت وتُعدل مع تغيير الوقت.
معطل	جرى تعطيل عمل الفرن لضرورات الأمان. تُظهر الشاشة "غير جاهز".
مفعل	جرى تفعيل عمل الفرن. الضوء داخل الفرن مُطفأ. تُظهر الشاشة " جاهز للطبخ".
تشغيل	الفرن يعمل. الضوء داخل الفرن يعمل. تُظهر الشاشة العد التنازلي للمؤقت.

	عند انتهاء الطبخ يطلق الفرن صفير لخمس ثواني. يُشعل ضوء الفرن. تُظهر الشاشة "انتهاء الطبخ" مع صوت الصفير.
الحدث	الوصف
نصف طاقة	ضغط المستخدم على زر"نصف طاقة".
طاقة كاملة	ضغط المستخدم على زر "طاقة كاملة".
المؤقت	ضغط المستخدم على أحد أزرار المؤقت.
رقم	ضغط المستخدم على رقم.
باب مفتوح	باب الفرن غير مغلق.
باب مغلق	باب الفرن مغلق.
بدء	ضغط المستخدم على زر البدء.
إلغاء	ضغط المستخدم على زر الإلغاء.



نماذج المعطيات

وهي النماذج التي تستخدم لوصف البنية المنطقية للمعطيات المستخدمة في النظام، منها نموذج الكيانات والعلاقات Entity Relationship Model الذي يحدد الكيانات المستخدمة في النظام مع واصفاتها والعلاقات فيما بينها. تُستخدم هذه النماذج لتصميم قواعد المعطيات ويمكن بسهولة تنجيزها بقواعد معطيات علاقاتية. يمكن استخدام صفوف الأغراض والارتباطات في UML للتعبير عن هذه النماذج. يُظهر الشكل التاي نموذج معطيات نظام المكتبة.



كما في جميع المخططات يحتاج المحللون إلى إضافة معلومات نصية فيستخدمون **قاموس المعطيات** وهو قائمة من الأسماء المستخدمة في النظام من كيانات وواصفات وعلاقات تسمح بإدارة المعطيات ومنع تكرار التسميات وربط معلومات التحليل والتصميم والتنجيز وتنظيمها كما في الجدول التالي (قاموس معطيات نظام المكتبة).

الاسم	الوصف	النوع	التاريخ
مقال	تفاصيل مقال منشور يمكن طلبه باستخدام نظام المكتبة.	كيان	2002/12/30
مؤلفون	أسماء مؤلفي المقال الذين يمكن أن يتشاركوا في ثمنه.	واصفة	2002/12/30
مشتري	الشخص أو المؤسسة التي يمكن أن تطلب نسخة من المقال.	كيان	2002/12/30
يُدفع الثمن إلى	علاقة 1:1 بين المقال ووكالة حقوق الطبع التي يجب دفع الثمن لها.	علاقة	2002/12/29
عنوان المشتري	عنوان المشتري الذي يستخدم عند طلب معلومات جباية الفواتير.	واصفة	2002/12/31

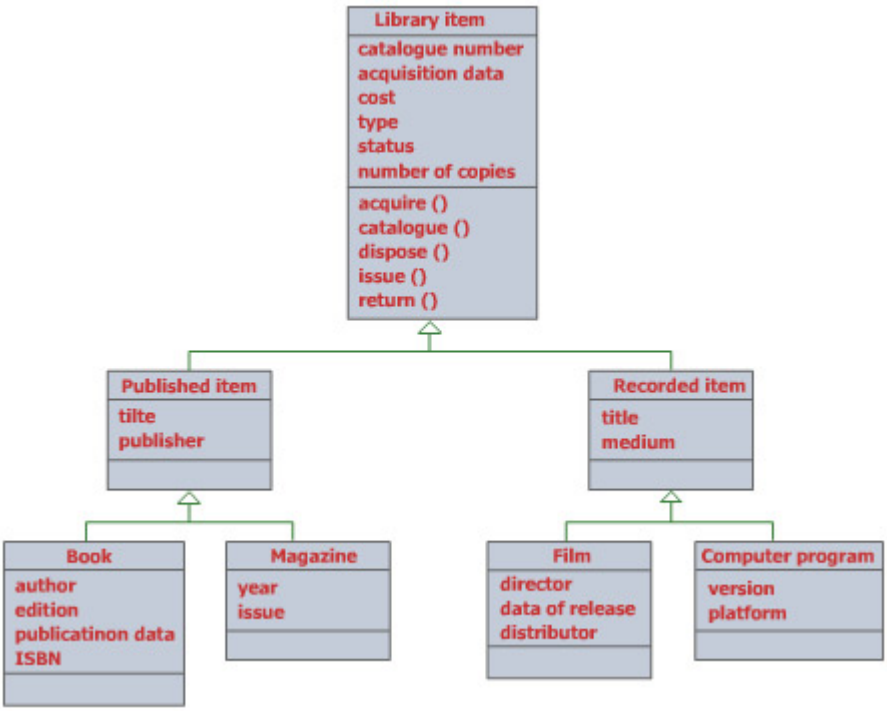
النماذج الغرضية

تصف هذه النماذج النظام عن طريق صفوف الأغراض وارتباطاتها وسلوكها. حيث يمثل كل صف مفهوم تجريدي لنوع من الأغراض التي لها واصفات مشتركة وتقدم نفس الخدمات (العمليات). تُعتبر هذه النماذج الطريقة الطبيعية لتمثيل معطيات العالم الحقيقي وتعبر عن فهم عميق لمجال التطبيق.

أصبحت لغة النمذجة الموحدة UML معياراً فعلياً للنمذجة الغرضية مستخدماً بكثرة في طرق التحليل والتصميم الغرضية التوجه. تتضمن هذه اللغة عدة أنواع من المخططات سنرى نماذج منها في الفقرات التالية:

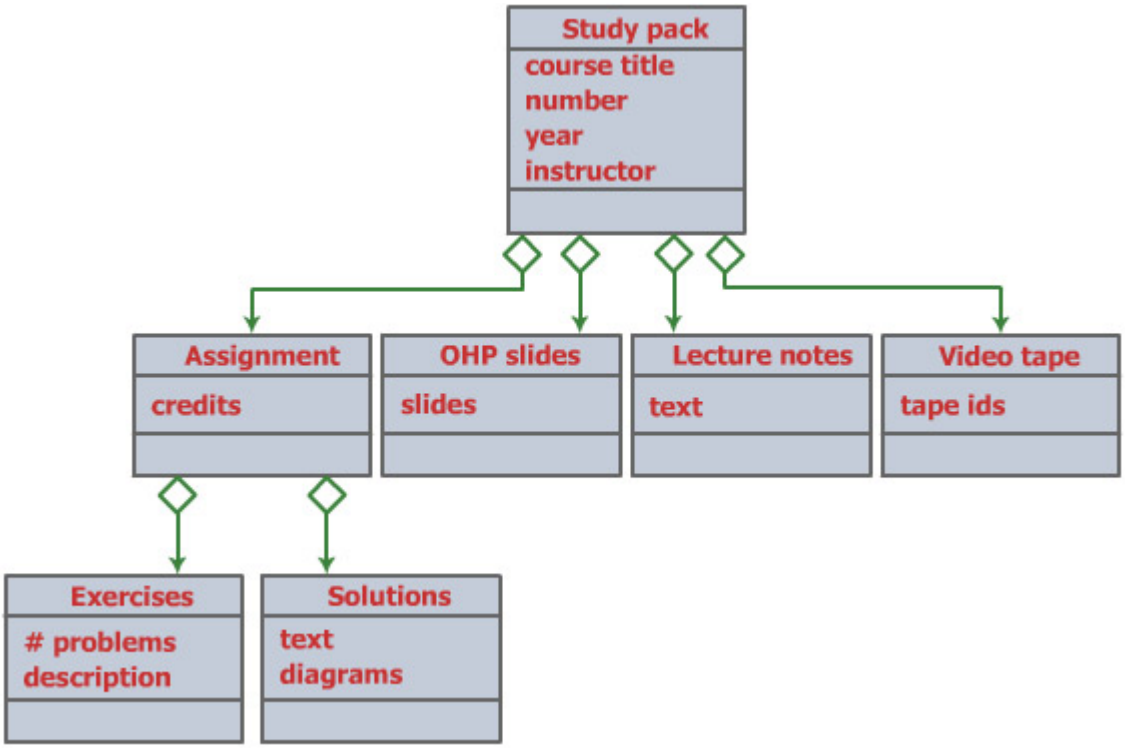
1- نماذج الوراثة

تُنظم صفوف الأغراض ضمن هرمية حيث تجمع الصفوف العليا المزايا المشتركة لجميع الصفوف وترث الصفوف الجزئية الواصفات والخدمات من الصفوف الرئيسية. يُظهر الشكل التالي مخطط الصفوف class diagram الخاص بنظام المكتبة.



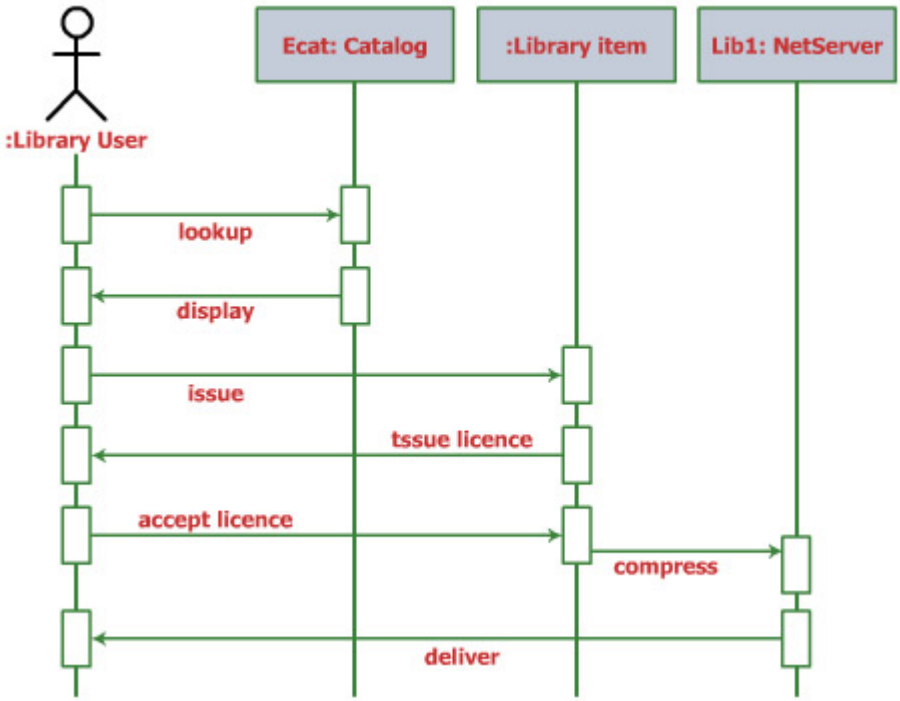
2- نماذج التجميع

تُظهر نماذج التجميع الصفوف المؤلفة من صفوف أخرى عن طريق روابط تجميع تشبه العلاقات في نماذج المعطيات. في الشكل التالي مخطط صفوف يُظهر نموذج تجميعي لمادة دراسية.



3- نماذج السلوك

تُظهر نماذج السلوك التفاعل بين الأغراض للتعبير عن سلوك معين في النظام جرى توصيفه في حالة استخدام. من المخططات المستخدمة في UML للتعبير عن هذه النماذج **مخططات التسلسل sequence diagrams** (مخططات التعاون collaboration diagrams). يُظهر الشكل التالي مخطط التسلسل الخاص بإصدار عناصر إلكترونية في نظام المكتبة.



5. توصيف الواجهات

تعمل معظم الأنظمة مع أنظمة أخرى مما يستوجب توصيف متطلبات واجهات التشغيل المتعلقة بها.
هنالك ثلاثة أنواع من الواجهات التي يمكن تعريفها:

1- الواجهات الإجرائية.

2- بنى المعطيات التي يجري تبادلها.

3- واجهات تمثيل وعرض المعطيات.

الطريقة الأكثر فعالية لتوصيف الواجهات هي التدوينات الصورية التي سنتعرض لها في الفصل الرابع. يمكن مثلاً توصيف واجهة إجرائية لمخدم طباعة باستخدام لغة Java بالشكل التالي:

```
interface PrintServer {
// defines an abstract printer server
interface Printer, interface PrintDoc     // requires:
// provides: initialize, print, displayPrintQueue, cancelPrintJob, switchPrinter
void initialize ( Printer p ) ;
void print ( Printer p, PrintDoc d ) ;
void displayPrintQueue ( Printer p ) ;
void cancelPrintJob (Printer p, PrintDoc d) ;
  void switchPrinter (Printer p1, Printer p2, PrintDoc d) ;
} //PrintServer
```

6. وثيقة المتطلبات البرمجية

وثيقة المتطلبات هي الوثيقة الرسمية التي تُبين المطلوب من مطوري النظام. يجب أن تتضمن هذه الوثيقة تعريفاً بمتطلبات المستخدم وتوصيفاً لمتطلبات النظام. ليست هذه الوثيقة تصميمية بمعنى أنها تحدد ما يجب أن يقوم النظام بعمله وليس كيف يجب أن يقوم بهذا العمل.

إن تنوع مستخدمي وثيقة المتطلبات يعني أنها يجب أن توفق بين وضوح المتطلبات للزبائن، وتعريفها بتفاصيل دقيقة للمطورين والمختبرين، وأن تحتوي معلومات عن التطورات الممكنة على النظام.

يتعلق مستوى التفصيل المطلوب بنوع النظام وإجرائية التطوير، فعند تطوير نظام حرج ذو أهمية كبير من قبل متعاقد خارجي يجب أن تكون المتطلبات شديدة التفصيل والدقة. أما عند وجود مرونة أكبر في المتطلبات واتباع منهجية تطوير تكرارية ضمن المؤسسة فيمكن أن تكون الوثيقة أقل تفصيلاً.

تعرف المؤسسات الكبيرة معايير خاصة بوثائق المتطلبات من أشهرها **معيار IEEE** الذي يقترح البنية التالية لوثيقة المتطلبات:

1- مقدمة

1- هدف وثيقة المتطلبات.

2- طيف المنتج.

3- تعريف، اختصارات، مصطلحات.

4- مراجع.

5- ملخص عن تتمة الوثيقة.

2- وصف عام

1- منظور المنتج.

2- وظائف المنتج.

3- ميزات المستخدمين.

4- قيود عامة.

5- فرضيات واعتماديات.

3- متطلبات تفصيلية

تمثل هذه المتطلبات الجزء الأساسي من الوثيقة لكن التنوع الكبير في ممارسات المؤسسات جعل من غير الممكن وضع بنية عامة لها فيمكن أن تتضمن حسب الحاجة الأجزاء التالية:

1- متطلبات وظيفية.

2- متطلبات غير وظيفية.

3- متطلبات توصيف الواجهات.

4- متطلبات التخزين.

5- قيود التصميم.

6- متطلبات نمو النظام.

7- معلومات الجودة.

مع أن المعيار IEEE ليس مثالياً إلا أنه قد يكون أساساً لبنية عامة لوثيقة المتطلبات يمكن تكييفها حسب النظام. يمكن أن نتوسع في هذا المعيار لوضع بنية أكثر تفصيلاً وتأخذ بعين الاعتبار توقعات تطور النظام بما يسمح للمصممين بدعم الميزات المستقبلية ويسهل صيانة النظام (انظر الجدول التالي: بنية موسعة لوثيقة المتطلبات).

الفقرة	الوصف
تقديم	القراء المتوقعين وتاريخ الإصدارات مع وصف التغييرات في كل منها.
مقدمة	الحاجة إلى النظام ووصف مختصر عن وظائفه وكيفية عمله مع باقي الأنظمة وموقعه ضمن أعمال المؤسسة وأهدافها الإستراتيجية.

تعاريف	التعابير التقنية المستخدمة في الوثيقة.
تعريف متطلبات المستخدم	الخدمات المقدمة للمستخدم والمتطلبات غير الوظيفية بلغة طبيعية أو مخططات مفهومة للزبائن مع تحديد لمعايير المنتجات والإجراءات.
بنية النظام	رؤية شاملة من المستوى الأعلى تُظهر توزع الوظائف على الكتل.
تحديد متطلبات النظام	المتطلبات الوظيفية وغير الوظيفية مفصلة وواجهات الربط مع الأنظمة.
نماذج النظام	نماذج المعطيات أو تدفق المعطيات أو نماذج غرضية تظهر علاقة النظام مع بيئته والعلاقات بين مكوناته.
تطور النظام	وصف للافتراضات التي يعتمد عليها النظام والتغييرات المحتملة بسبب تطور العتاد المادي أو تغير حاجات المستخدمين.
ملحقات	معلومات تفصيلية محددة تتعلق بالتطبيق كوصف العتاد المادي أو قاعدة المعطيات.
الفهارس	مجموعة من الفهارس الأبجدية، فهارس المخططات، الجداول ...

1. مقدمة

سبق أن اطلعت في مقرر "تحليل وتصميم النظم" على بعض التقنيات والطرائق والأدوات المستخدمة في توصيف البرمجيات. في هذا المقرر سنركز على نوع خاص من تقنيات التوصيف المعتمدة على الرياضيات، وسنبين مزاياها ومدى انتشارها.

لقد أدت الأبحاث والتطورات الحاصلة في الاختصاصات الهندسية التقليدية كالهندسة المدنية والإلكترونية إلى تطوير تقنيات رياضية اعتبرت جزءاً من إجرائية التطوير (كاستخدام التحليل الرياضي في تطوير المنتج والتحقق من صلاحيته).

فكيف هو الأمر في استخدام هذه التقنيات في تطوير البرمجيات؟ وهل لاقت هذه التقنيات في هندسة البرمجيات النجاح نفسه وأصبحت جزءاً من إجرائية التطوير؟

سنتعرف أولاً معنى الطرائق الصورية ثم نركز على دورها في التطوير.

2. الطرائق الصورية

يعتبر التوصيف الصوري formal specification جزءاً من مجموعة أكبر من التقنيات تعرف **باسم الطرائق الصورية formal methods**. تستند الطرائق الصورية إلى تمثيل رياضي للبرمجية، وهي تشمل عدة تقنيات نذكر منها:

1- التوصيف الصوري[1] formal specification.

2- تحليل المواصفات والبرهنة عليها specification analysis and proof.

3- التطوير التحويلي transformational development.

4- التحقق من صحة البرامج program verification.

لم يحصد استخدام هذه التقنيات النجاح نفسه الذي حصده في الاختصاصات الهندسية الأخرى، فرغم كثرة الأبحاث على مدى الأعوام الثلاثين الماضية بهدف اعتماد التقنيات الرياضية كجزء من الإجرائية البرمجية، إلا أنها بقيت محدودة التأثير، ولم تستخدم الطرائق الصورية استخداماً كافياً في صناعة تطوير البرمجيات. يعزو البعض ذلك إلى الأسباب التالية:

1- تمكنت التقنيات الأخرى في هندسة البرمجيات من إثبات نجاحها في زيادة جودة النظم البرمجية، وبذلك تراجعت الحاجة إلى الطرائق الصورية.

[1] نقصد بالتوصيف الصوري للبرمجية توصيفاً تمت كتابته بلغة صورية، بمعنى أن مفردات هذه اللغة ونَحوها ودلالتها معرفة تعريفاً صورياً. ويعني التعريف الصوري أن اللغة مبنية على مفاهيم رياضية واضحة الخصائص. تستخدم في هذا المجال الرياضيات المتقطعة discrete mathematics ونظرية المجموعات والمنطق والجبر.

2- أعطت التغييرات الطارئة على سوق البرمجيات الأولوية لسرعة وضع البرمجية في السوق بدلاً من انخفاض عدد الأخطاء فيها، ولا تقدم الطرائق الصورية أي تسريع في وضع البرمجية في السوق.

3- إن استخدام الطرائق الصورية محدود، إذ أنها لا تناسب تحليل وتوصيف واجهات المستخدم وتفاعله.

4- لا تزال الطرائق الصورية صعبة التطبيق في الأنظمة كبيرة الحجم.

ما الذي قدمته إذاً الطرائق الصورية لإجرائية التطوير البرمجي؟ وأين كانت نتائجها المبهرة؟

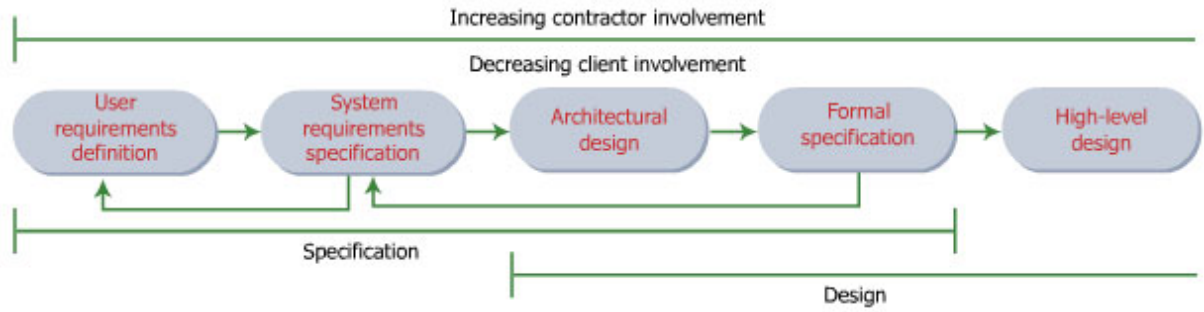
لقد ظهرت الفائدة الأساسية للطرائق الصورية في تخفيض عدد الأخطاء في النظام، مما أدى إلى أن يكون للطرائق الصورية دوراً أساسياً في **هندسة الأنظمة الحرجة** critical systems التي تتطلب درجة عالية من السلامة safety والموثوقية reliability والأمن security. يؤدي استخدام الطرائق الصورية في مجال الأنظمة الحرجة إلى توفير للكلف لأنه يجنّب الكلف العالية الناتجة عن الأخطاء ومعالجتها. وقد استعملت هذه الطرائق في العديد من المشاريع الناجحة نذكر منها نظم المعلومات الخاصة بالمراقبة الجوية، ونظم التأشير الخاصة بالسكك الحديدية، ونظم المركبات الفضائية، ونظم المراقبة الطبية.

3. التوصيف الصوري في الإجرائية البرمجية

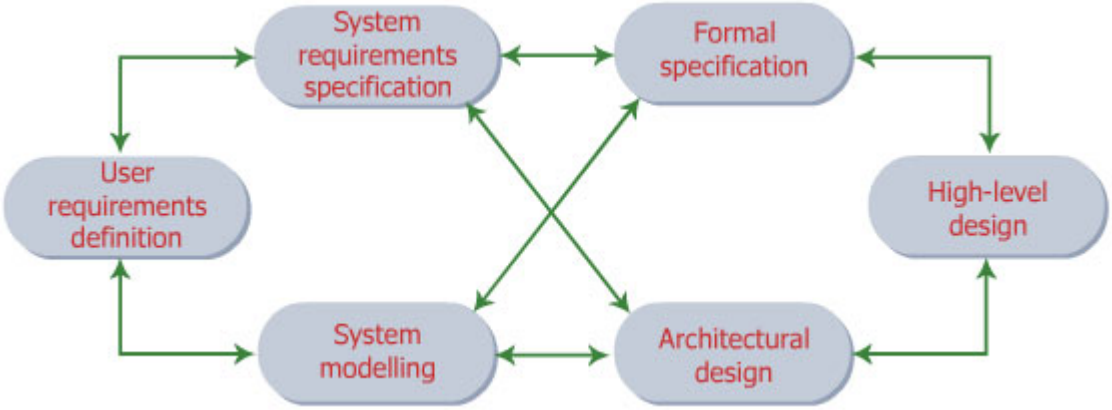
تطور النظم الحرجة عادةً باستخدام النموذج الشلالي (الفصل الأول الفقرة 5.1.1). وكما رأينا سابقاً، يجب في هذا النموذج أن يجري التعبير عن مواصفات النظام كاملة، ثم تصميمه بالتفصيل، ويتم تحليل المواصفات والتصميم والتحقق منهما قبل البدء بالتنجيز. يجري العمل على التوصيف الصوري للبرمجية بعد توصيف متطلبات النظام وقبل تصميمه التفصيلي. توجد حلقة ارتباط قوية بين التوصيف التفصيلي للمتطلبات والتوصيف الصوري، في الواقع يكشف التوصيف الصوري جميع الالتباسات ambiguities الواردة في متطلبات النظام.

مع الدخول أكثر فأكثر في المواصفات التفصيلية للنظام تقل مشاركة الزبون وتزداد مشاركة الشركة المنفذة. ففي البدء يجب أن يكتب التوصيف بحيث يفهمه الزبون مع تجنب الإشارة إلى أية إيحاءات تتعلق بالنظام. أما في نهاية مرحلة التوصيف التي تهدف إلى وضع مجموعة مكتملة ودقيقة ومتجانسة من المواصفات فيجب أن تكتب المواصفات بحيث يفهمها مهندس البرمجيات، وهنا يمكن استخدام لغة صورية للتعبير عن المواصفات بشكل يخلو من الخطأ والالتباس.

يبين الشكل مراحل توصيف البرمجيات وعلاقتها بالتصميم.

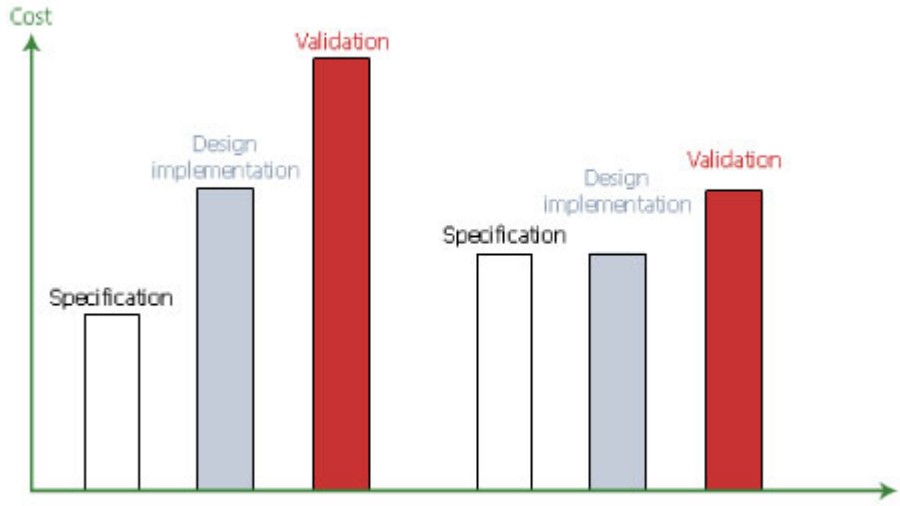


كما يبين الشكل أنه يمكن القيام بنشاطي التوصيف والتصميم ضمن مسارين متوازيين.



يتطلب استخدام التوصيف الصوري بذل المزيد من الجهد في المراحل الأولى من التطوير البرمجي، إذ أنه يجبر المهندس على إجراء تحليل تفصيلي للمتطلبات مما يخفّض من الأخطاء الناتجة عن المتطلبات. ونذكّر هنا بأنه كثيراً ما تكون المتطلبات التي يضعها الزبون غير مكتملة incomplete وغير متجانسة inconsistent، فيقوم التوصيف الصوري بالكشف عن هذه الإشكالات وحلها.

إن استخدام التوصيف الصوري يساهم في توفير الكلف، لأن المعالجة المبكرة للنقص وعدم التجانس والالتباس الوارد في متطلبات الزبون يجنّب إعادة العمل بسبب الأخطاء الموجودة في المتطلبات، وهذا ما يؤدي بالتالي إلى تغيير في المنحني المتعارف عليه لكلفة المشروع، حيث تنفق كلف كبيرة في البداية بسبب الجهد والوقت الإضافيين اللذين يصرفان في تطوير التوصيف، في حين تكون كلف التنجيز والتحقق من الصلاحية منخفضة. يبين الشكل التالي منحني توزع الكلف في حالة التطوير باستخدام التوصيف الصوري.



تقنيات التوصيف الصوري

استخدمت حتى الآن تقنيتان من تقنيات التوصيف الصوري وهما:

1- التوصيف الجبري Algebraic specification: في هذه التقنية يتم توصيف النظام اعتماداً على عملياته وعلى العلاقات فيما بينها.

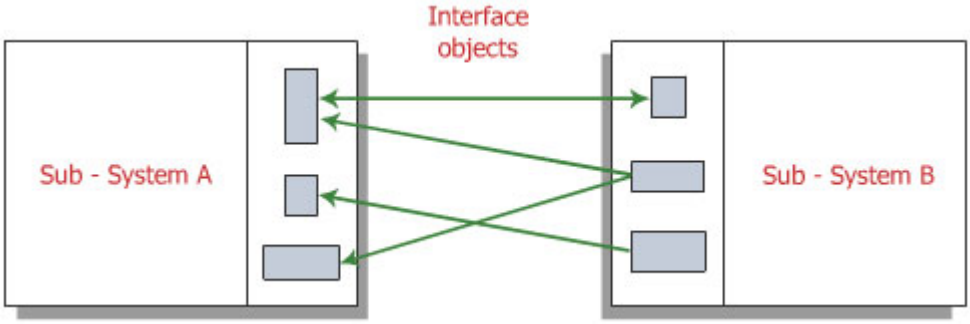
2- التوصيف المعتمد على النموذج Model-based specification: في هذه التقنية يتم توصيف النظام اعتماداً على نموذج للحالة يجري بناؤه باستخدام بنى رياضية كالمجموعات والمتتاليات، أما العمليات فتعرف بالتغييرات التي تحصل على حالة النظام.

لقد جرى تطوير عدد من لغات التوصيف الصوري، بعضها لتوصيف النظم التسلسلية وبعضها لتوصيف النظم التسايرية. يبين الجدول التالي أمثلة عن هذه اللغات:

لغات التوصيف الصوري	تسلسلي	تسايري
لغات جبرية	Larch OBJ	Lotos
لغات معتمدة على النموذج	Z VDM B	CSP شبكات بتري Petri Nets

1- توصيف الواجهات

تتألف الأنظمة الكبيرة من عدة أنظمة فرعية توجد بينها واجهات معرفة بوضوح، ويسمح توصيف الواجهات الخاصة بالأنظمة الفرعية بتطوير الأنظمة الفرعية كل على حده وبشكل مستقل. تعرف الواجهات **كأنماط معطيات مجردة abstract data type** أو كصفوف (أغراض) أو كمكونات انظر الشكل التالي. تناسب المقاربة الجبرية التي تُستخدم في التوصيف الصوري توصيف الواجهات نظراً لأنها تركّز على العمليات المعرفة في كل غرض.



بنية التوصيف الجبري

في المقاربة الجبرية للتوصيف الصوري تستخدم أنماط المعطيات المجردة ويعرف النمط (النوع) بدلالة العلاقات بين عمليات النمط المجرد. يتألف التوصيف من أربعة أجزاء (انظر الشكل التالي):

1- **المقدمة**: وتتضمن اسم النوع sort (أي اسم النمط المجرد) وتصريحاً عن التوصيفات الأخرى المستخدمة.

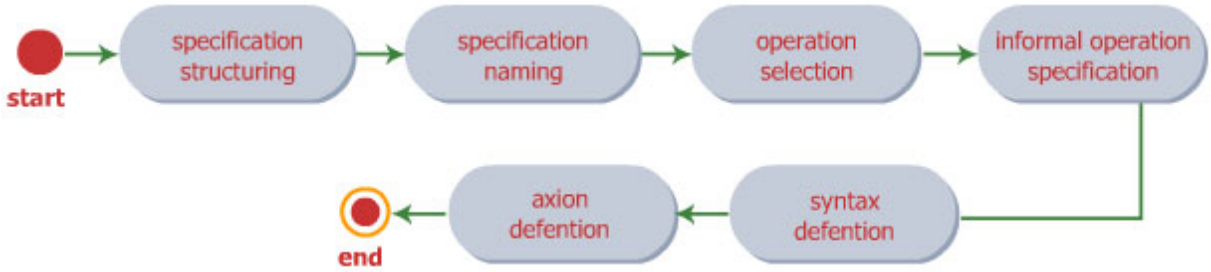
2- **التوصيف غير الصوري**: توصف فيه العمليات وصفاً غير صوري (أي وصفاً نصياً لا يتبع أية قواعد رياضية).

3- **التواقيع**: وتتضمن ترويسات العمليات وموسطاتها.

4- **المسلّمات**: وتتضمن معاني العمليات عبر وضع عدد من المسلّمات axiom التي تصف سلوك النمط المجرد.

إجرائية التوصيف الجبري

تعتمد إجرائية تطوير التوصيف الصوري لواجهة نظام فرعي على القيام بالأنشطة التالية:



1- **إعطاء بنية للتوصيف specification structuring**: يجري في هذه المرحلة تنظيم التوصيف غير الصوري للواجهة ضمن مجموعة من أنماط المعطيات المجردة أو صنوف الأغراض.

2- **تسمية التوصيف specification naming**: يجري إعطاء اسم لكل نمط مجرد.

3- **اختيار العمليات operation selection**: يجري اختيار مجموعة عمليات لكل توصيف اعتمادا على الوظائف التي تم تحديدها للواجهة.

4- **التوصيف غير الصوري للعمليات informal operation specification** : تجري كتابة توصيف غير صوري لكل عملية يبين كيف تؤثر العملية على النوع sort.

5- **تعريف النحو syntax definition**: يجري تعريف النحو الخاص بكل عملية وموسطاتها.

6- **تعريف المسلّمات axiom definition**: يجري تعريف معنى كل عملية عبر وصف الشروط الصحيحة لتركيب العمليات بعضها مع بعض.

عمليات التوصيف

تصنف العمليات الخاصة بالأنماط المجردة ضمن صنفين:

1- **العمليات البوانية Constructor operations**: وهي العمليات التي تنشئ أو تعدل الكيانات entities المأخوذة من النوع sort المعرف في الواجهة. ويكون اسمها عادة Create, Update, Add, Cons,...

2- **عمليات التحري Inspection operations:** وهي العمليات التي تحسب قيمة واصفات النمط المعرف. ويكون اسمها عادة Eval, Get,...

ولتوصيف السلوك، يجب أن تعرف عمليات التحري الخاصة بكل عملية باني.

3. التوصيف الصوري في الإجرائية البرمجية

مثال

لنأخذ نمط المعطيات المجرد List. يمثل هذا النمط لائحة من العناصر المرتبة بحيث يمتلك كل عنصر فيها وصلة تصله بالعنصر الذي يليه. نفترض هنا أن المرحلة الأولى من إجرائية التوصيف وهي "إعطاء بنية للتوصيف" قد أسفرت عن الحاجة إلى نمط "لائحة". يمكن أن نسمي الوصيف والنوع

بالاسم ذاته، مع استخدام الأحرف الكبيرة لاسم التوصيف **LIST** والصغيرة لاسم النوع **list**. تتألف اللائحة من جمع من الأنماط الأخرى، لذا نعطي للتوصيف موسطاً عمومياً نسميه **Elem**. يمثل **Elem** عنصراً من نمط صحيح أو سلسلة محارف أو ربما لائحة. العمليات التي تعرف على **List** هي:

1- **العمليات البوانية**: **Create** لإنشاء منتسخات من العدم من هذا النوع (أي لوائح جديدة)، و **Cons** لإنشاء لائحة انطلاقاً من عناصرها الأساسية، و **Tail** وهي عملية تعيد اللائحة المتبقية بعد إزالة العنصر الموجود في رأس اللائحة.

2- **عمليات التحري**: **Head** وهي تعيد العنصر الموجود في رأس اللائحة، و **Length** هي تعيد عدد العناصر الموجودة في اللائحة.

تفويم: لماذا كانت **Head** و **Length** عمليات تحري وليست بوانية؟

الحل

لأن البواني تعطي نتيجة عملها كياناً من النوع نفسه، أما العمليات Head و Length فهي تأخذ اللائحة كموسط وتعيد نتيجة من نوع آخر (Head تعيد عنصراً Elem و Length تعيد عدداً صحيحاً).

لتحديد النحو الخاص بكل عملية يجب تحديد موسطات الدخل ونوع النتيجة التي تعيدها. عموماً، تكون الموسطات إما من النوع **List** الذي يجري تعريفه، أو من نمط العنصر العمومي **Elem**، وتكون نتائج العمليات إما من النوع نفسه أو نتيجة صحيحة أو منطقية. لذا يجب أن نضع في فقرة **imports** تصريحاً عن استيراد النمط **INTEGER**، كما نبين أنماط دخل وخرج العمليات **Create**, **Cons**, **Tail**, **Head**, **Length**.

أما معاني العمليات فيجري التعبير عنها بمجموعة مسلّمات. تكتب هذه المسلّمات بدلالة تواقيع (ترويسات) التوابع التي كتبت في الفقرة السابقة. وقد تُكتب بعض التوابع بدلالة بعضها الآخر (كما يكتب Tail بدلالة Create و Cons)، وكثيراً ما تُعرف بعض التوابع بطريقة **عودية** **recursive** كما فيما يلي:

Tail (Cons (L, v)) = if L = Create then Create
else Cons (Tail (L), v).

لفهم التوصيفات العودية يفضل كتابة أمثلة تطبيقية، كما في الأمثلة التالية:

Cons ([5, 7], 9) = [5, 7, 9]
Tail ([5, 7, 9]) = Tail (Cons ([5, 7], 9))
= Cons (Tail ([5, 7]), 9)
= Cons (Tail (Cons ([5], 7)), 9)
= Cons (Cons (Tail ([5]), 7), 9)

= Cons (Cons (Tail (Cons ([ ], 5)), 7), 9)
= Cons (Cons ([Create], 7), 9)
= Cons ([7], 9)
= [7, 9]

التعريف الصوري للنمط قائمة List:

LIST (Elem)

sort List
imports INTEGER

Defines a list where elements are added at the end and removed from the front.The operations are Create, which brings an empty list into existence, Cons, which creates a new list with an added member, Length, whiche valuates the list size, Head, whiche valuates the front element of the list, and Tail, which creates a list by removing the head from its input list. Undefined represents an undefined value of type Elem.

Create → List
Cons (List , Elem) → List
Head (List) → Elem
Length (List) → Integer
Tail (List) → List

Head (Create) = Undefined **exception** (empty list)
Head (Cons (L, v)) = **if** L = Create **then** v else **Head** (L)
Length (Create) = 0
Length (Cons (L, v)) = Leng th (L) + 1
Tail (Create) = Create
Tail (Cons (L, v)) = **if** L = Create **then** Create **else** Cons (Tail (L), v)

تدريب

اكتب مثالاً تتحقق فيه من المسلّمة التي عرفنا بها التابع Head كما فعلنا في حالة التابع Tail.

توصيف الواجهة في النظم الحرجة

في هذه الفقرة سترى كيف يمكن استخدام التوصيف الجبري لتوصيف واجهة لنظام حرج. لنفترض أننا بصدد توصيف نظام تحكم بحركة الطائرات، حيث تطير الطائرات ضمن قطاعات sectors محددة يمكن إدارتها. قد يحتوي كل قطاع عدداً من الطائرات لكن وبدافع الأمان يجب أن تنفصل هذه القطاعات بعضها عن بعض. في هذا المثال سنعتبر أن الفاصل بين القطاعات هو فاصل عمودي بمقدار 300م. يجب أن يحذر النظام المراقبَ إذا ما كانت تعليمات الطائرات تسمح لها باختراق هذا الفاصل.

الغرض sector: تحتوي الواجهة على النوع sector ويمثل القطاع تحت المراقبة. تعرف على هذا النوع العمليات الحرجة التالية:

1- Enter: وهي تضيف طائرة إلى القطاع تحت المراقبة.
2- Leave: وهي تنزع طائرة من القطاع تحت المراقبة.
3- Move: وهي تنقل طائرة من ارتفاع إلى آخر.

4- Lookup: وهي تبحث عن طائرة ذات رقم معرف محدد وتعيد ارتفاعها الحالي.

من المفيد في كثير من الأحيان أن نضيف بعض التوابع لتبسيط التوصيف. عندها، تكتب التوابع الأخرى بدلالة هذه التوابع البدائية.

في مثالنا هذا يمكن أن تكون التوابع البدائية هي:

1- Create: ينشئ منتسخاً من القطاع.

2- Put: يضيف طائرة دون التحقق من شروط الأمان.

3- In-space: يحدّد ما إذا كانت طائرة معطاة موجودة في القطاع.

4- Occupied: يعطى هذا التابع ارتفاعاً محدداً، فيقوم بتحديد ما إذا كانت هناك طائرة ضمن مجال 300 م من هذا الارتفاع.

نلاحظ في هذه التوابع النقاط التالية:

1- إن التابعين Create و Put هما البانيان الأساسيان، ويستخدمان في توصيف التوابع الأخرى.

2- جرى تعريف توابـع التفقّـد Occupied و In-space باستخدام Create و Put ثم جرى استخدامهما في توصيف التوابع الأخرى.

3- يجب على جميع التوابع التي تؤدي إلى تغيير في القطاع أن تتحقق من استمرار توفر شروط الأمان.

يبين **الشكل التالي** التوصيف الصوري للقطاع.

SECTOR

sort Sector
import INTEGER, BOOLEAN

Enter - adds an aircraft to the sector if safety conditions are satisfied
Leave - removes an aircraft from the sector
Move - moves an aircraft from one height to another if safe to do so
Lookup - Finds the height of an aircraft in the sector

Create - creates an empty sector
Put - adds an aircraft to a sector with no constraint checks
In-space - checks if an aircraft is already in a sector
Occupied - checks if a specified height is available

Enter (Sector , Call-sign, Height) → Sector
Leave (Sector , Call-sign) → Sector
Move (Sector , Call-sign, Height) → Sector
Lookup (Sector , Call-sign) → Height
Create → Sector
Put (Sector , Call-sign, Height) → Sector
In-space (Sector , Call-sign) → Boolean
Occupied (Sector , Height) → Boolean

Enter (S, CS, H) =
  if In-space (S, CS) **then** S **exception** (Aircraft already in sector)
  elsif Occupied (S, H) **then** S **exception** (Height conflict)
  else Put (S, CS, H)

Leave (Create, CS) = Create **exception** (Aircraft not in sector)
Leave (Put (S, CS1, H1), CS) =
  if CS = CS1 **then** S else Put (Leave (S, CS), CS1, H1)

Move (S, CS, H) =
  if S = Create **then** Create **exception** (No aircraft in sector)
  elsif not In-space (S, CS) **then** S **exception** (Aircraft not in sector)
  elsif Occupied (S, H) **then** S **exception** (Height conflict)
  else Put (Leave (S, CS), CS, H)

--NO - EIGHT is a constant indicating that a valid height cannot be returned

Lookup (Create, CS) = NO - EIGHT **exception** (Aircraft not in sector)
Lookup (Put (S, CS1, H1), CS) =
  if CS = CS1 **then** H1 **else** Lookup (S, CS)

Occupied (Create, H) = false
Occupied (Put (S, CS1, H1), H) =
  if (H1 > H **and** H1 - H ≤ 300) **or** (H > H1 **and** H - H1 ≤ 300) **then** true
  else Occupied (S, H)

In-space (Create, CS) = false
In-space (Put (S, CS1, H1), CS) =
  if CS = CS1 **then** true **else** In-space (S, CS)

2- التوصيف السلوكي Behavioral specification

قد يكون التوصيف الجبري ثقيلاً ومزعجاً عندما لا تكون عمليات الغرض مستقلة عن حالته. **وتكون عمليات الغرض مستقلة عن حالته عندما لا تتأثر نتائج تنفيذ عملية ما بالعمليات السابقة لها**. غير أن عدم توفر هذا الاستقلال في بعض الحالات يجعل من المزعج استخدام التوصيف الجبري. في مثل هذه الحالات يمكن اللجوء إلى التوصيف المعتمد على النموذج model-based specification، وهو توصيف يعرض حالة النظام ويعرف العمليات بدلالة التغيرات التي تطرأ على هذه الحالة.

توجد العديد من التدوينات واللغات الناضجة والمستخدمة في التوصيف المعتمد على النموذج مثل VDM و B و Z. وسنعرض لغة Z مثالاً عنها.

3. التوصيف الصوري في الإجرائية البرمجية

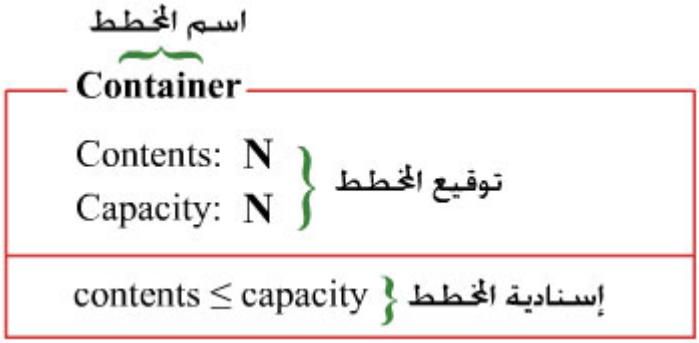
2- التوصيف السلوكي Behavioral specification

نستطيع في لغة Z نمذجة الأنظمة باستخدام المجموعات والعلاقات بين المجموعات، إضافة إلى أن هذه اللغة تمتلك بنى خاصة لتوصيف الأنظمة.

وقد تنبه مصممو لغة Z إلى صعوبة قراءة التوصيف المكتوب بطريقة صورية، لذا عمدوا إلى جعل التوصيف يكتب على شكل نص غير صوري مزود بشروح صورية. يصاغ هذا الشرح الصوري على شكل كتل (تسمى مخططات schemas) يمكن تمييزها عن النص نظراً لأنها تُعرَض بطرقة بيانية كما يبين الشكل التالي. تعرف المخططات ما يسمى بمتحولات الحالة، وتتضمن العمليات والقيود على هذه المتحولات.

بنية المخطط في لغة Z

لا يتسع المجال هنا لعرض لغة Z كاملة، لكننا سنستعرض كيفية استخدامها في التوصيف، وسنستخدم لهذا الغرض مثالاً لتوصيف نظام حرج، هو نظام التحكم بمضخة أنسولين.



نظام التحكم بمضخة الأنسولين

راجع الفقرة "4- متطلبات النظام: 4-1- كتابة المتطلبات بلغة بنيوية" من الفصل الثالث حيث توجد متطلبات المسألة.

يراقب هذا النظام مستوى السكر في دم المريض، وعند اللزوم يقوم آلياً بحقن المريض بكمية من الأنسولين. رغم أن النظام يبدو صغيراً، إلا أن توصيفه الصوري سيكون طويلاً. كما أنه وإن كانت العملية الأساسية للنظام بسيطة إلا أنه يوجد عدد من شروط الإنذار التي يجب الاهتمام بها.

لتطوير توصيف معتمد على النموذج يجب أن نحدد متحولات الحالة state variables والإسناديات predicates التي تنمذج حالة النظام إضافة إلى تعريف اللامتغيرات (وهي الشروط التي تكون دوماً صحيحة).

متحولات الحالة state variables

في مخطط Z الذي ينمذج حالة النظام يتم التصريح عن عدد من متحولات الحالة .

ومنها:

1- **متحولات الدخل**: ويعبر عنها في Z باسم المتحول يليه الرمز(؟)، مثل المتحول **switch?** (قاطعة الوصل/الفصل switch)، و **InsulinReservoir?** (الكمية الحالية من الأنسولين في المخزن)، و **Reading?** (القراءة من المحِس).

2- **متحولات الخرج**: ويعبر عنها في Z باسم المتحول يليه الرمز **(!)**، مثل **alarm!** (إنذار النظام) **display1!** و **display2!** (المِظهارات الحرفية الموجودة على المضخة) و **dose!** (جرعة الأنسولين التي يجب حقنها).

3- **متحولات الحالة المستخدمة في حساب الجرعة**: ونذكر منها **status** (لحالة الجهاز) و **r0** و **r1** و **r2** (لتخزين قيم سكر الدم المقاسة سابقاً) وغيرها.

الإسناديات predicates

تعرف الإسناديات في مخطط Z اللامتغيرات invariants التي تعبر عن الشروط التي تكون صحيحة دوماً.

لنبحث في مثالنا عن هذه الشروط. لدينا دوماً الشروط الصحيحة التالية:

1- يجب أن تكون جرعة الأنسولين أقل أو تساوي سعة مخزن الأنسولين.

2- يجب أن لا تتجاوز الجرعة الواحدة 4 وحدات من الأنسولين، كما يجب أن لا يتجاوز مجموع الجرعات المعطاة في اليوم 25 وحدة. وهذا الشرط هو شرط أمان.

3- يجري تصفير عداد الجرعات اليومية عند منتصف الليل كل يوم.

4- يبين المظهر display2! كمية الأنسولين الواجب تسليمها.

وفيما يلي مخطط Z الخاص بمضخة الأنسولين.

```
INSULIN_PUMP_STATE

//Input device definition

switch?: (off, manual, auto)
ManualDeliveryButton?: ℕ
Reading?: ℕ
HardwareTest?: (OK, batterylow, pumpfail, sensorfail, deliveryfail)
InsulinReservoir?: (present, notpresent)
Needle?: (present, notpresent)
clock?: TIME

//Output device definition
alarm! = (on, off)
display1!, string
display2!: string
clock!: TIME
dose!: ℕ
```

// State variables used for dose computation
status: (running, warning, error)
r0, r1, r2: ℕ
capacity, insulin_available : ℕ
max_daily_dose, max_single_dose, minimum_dose: ℕ
safemin, safemax: ℕ
CompDose, cumulative_dose: ℕ

---

r2 = Reading?
dose! ≤ insulin_available
insulin_available ≤ capacity

// The cumulative dose of insulin delivered is set to zero once every 24 hours
clock? = 000000 ⇒ cumulative_dose = 0

// If the cumulative dose exceeds the limit then operation is suspended
cumulative_dose ≥ max_daily_dose ∧ status = error ∧
display1! = “Daily dose exceeded”

// Pump configuration parameters
capacity = 100 ∧ safemin = 6 ∧ safemax = 14
max_daily_dose = 25 ∧ max_single_dose = 4 ∧ minimum_dose = 1

display2! = nat_to_string (dose!)
clock! = clock?

حساب الجرعات

تحسب مضخة الأنسولين كمية الأنسولين اللازمة عبر مقارنة القراءة الحالية بالقيم الموافقة لقراءتين سابقتين. إذا دلّت هذه القراءات على ارتفاع في سكر الدم، يجب عندها أن يضخ الأنسولين، ويحتَفظ بمعلومات عن مجموع الكميات التي تم ضخها ليصار إلى تطبيق لا متغير التحقق من الأمان. من الملاحظ أنه يمكن تطبيق هذا اللا متغير، ولا حاجة لتكراره عند حساب الجرعات.

تعمل المضخة بحيث تقوم باختبار سكر الدم كل 10 دقائق ويحقن الأنسولين إذا تبين أن نسبة تغير سكر الدم آخذة في الارتفاع. ينمذج المخطط **RUN** شروط العمل الطبيعية للمضخة. نلاحظ هنا أننا قمنا بإدراج اسم مخطط في الجزء الخاص بالتصريح مسبوقاً بالرمز (Δ)، وهذا يعني أننا نضمن جميع الأسماء المصرح عنها في ذلك المخطط ضمن المخطط الجديد.

RUN
ΔINSULIN_PUMP_STATE

---

switch? = auto
status = running ∨ status = warning
insulin_available ≥ max_single_dose
cumulative_dose < max_daily_dose
// The dose of insulin is computed depending on the blood sugar level

(SUGAR_LOW ∨ SUGAR_OK ∨ SUGAR_HIGH)
// 1. If the computed insulin dose is zero, don’t deliver any insulin
CompDose = 0 ⇒ dose! = 0

∨
// 2. The maximum daily dose would be exceeded if the computed dose was delivered so the insulin dose is set to the difference between the maximum allowed daily dose and the cumulative dose delivered so far

CompDose + cumulative_dose > max_daily_dose ⇒ alarm! = on ∧ status' = warning ∧ dose! = max_daily_dose – cumulative_dose

∨

// 3. The normal situation. If maximum single dose is not exceeded then deliver the computed dose. If the single dose computed is too high, restrict the dose delivered to the maximum single dose

CompDose + cumulative_dose < max_daily_dose ⇒
(CompDose ≤ max_single_dose ⇒ dose! = CompDose
∨
CompDose > max_single_dose ⇒ dose ! = max_single_dose)
insulin_available' = insulin_available – dose !
cumulative_dose' = cumulative_dose + dose !

insulin_available ≤ max_single_dose * 4 ⇒ status' = warning ∧
display1! = "Insulin low"

r1' = r2
r0' = r1

يعرف المخطط RUN كيفية عمل النظام من خلال تحديد مجموعة إسناديات، وتضاف إليها بالطبع الإسناديات الموجودة في المخطط INSULIN_PUMP_STATE.

يبين المخطط التالي كيفية حساب جرعة الأنسولين بافتراض أن مستوى السكر في دم المريض لا يزال ضمن مجال الأمان.

SUGAR_OK
r2 ≥ safemin ∧ r2 ≤ safemax
// sugar level stable or falling
r2 ≤ r1 ⇒ CompDose = 0
∨
// sugar level increasing but rate of increase falling
r2 > r1 ∧ (r2-r1) < (r1-r0) ⇒ CompDose = 0
∨
// sugar level increasing and rate of increase increasing compute dose
// a minimum dose must be delivered if rounded to zero
r2 > r1 ∧ (r2-r1) ≥ (r1-r0) ∧ (round ((r2-r1)/4) = 0) ⇒
CompDose = minimum_dose
∨
r2 > r1 ∧ (r2-r1) ≥ (r1-r0) ∧ (round ((r2-r1)/4) > 0) ⇒

الفصل الخامس جودة البرمجيات
Software Quality

1. إدارة جودة البرمجيات

يستخدم هذا التعبير للحديث عن الإجرائيات التي تهتم بضمان المستوى المطلوب من الجودة في منتج برمجي معين. يتضمن ذلك تعريف معايير وإجراءات جودة مناسبة وضمان تطبيقها. كما يجب أن يكون من أهدافه نشر ثقافة الجودة وتعميمها كمسؤولية لجميع الأطراف المشاركة في العمل البرمجي.

1. إدارة جودة البرمجيات

1 - ما هي الجودة؟

هي ببساطة أن المنتج يجب أن يُطابق التوصيف.

مع هذه البساطة تُعتبر الجودة موضع خلاف في الأنظمة البرمجية، فالتوصيفات نفسها تكون عادةً غير كاملة وغالباً غير متجانسة، وهنالك دوماً تنافس بين متطلبات الجودة من وجهة نظر الزبون التي تتمثل في الأداء العالي والوثوقية والمتطلبات غير الوظيفية الأخرى، وبين متطلبات الجودة من وجهة نظر المطور التي تتمثل في قابلية الصيانة وإعادة الاستخدام، ومتطلبات أخرى. عدا عن أن بعض متطلبات الجودة من الصعب توصيفها بطريقة واضحة.

في معظم الأحيان يكون الحل هو توفيق ما بين الجودة والتوصيف الموجود، فلا يمكننا الانتظار إلى حين تحسين التوصيفات دون إيلاء الاهتمام لإدارة الجودة. بل يجب أن نضع إجراءات إدارة الجودة موضع التنفيذ لتحسين الجودة رغم عدم اكتمال التوصيفات.

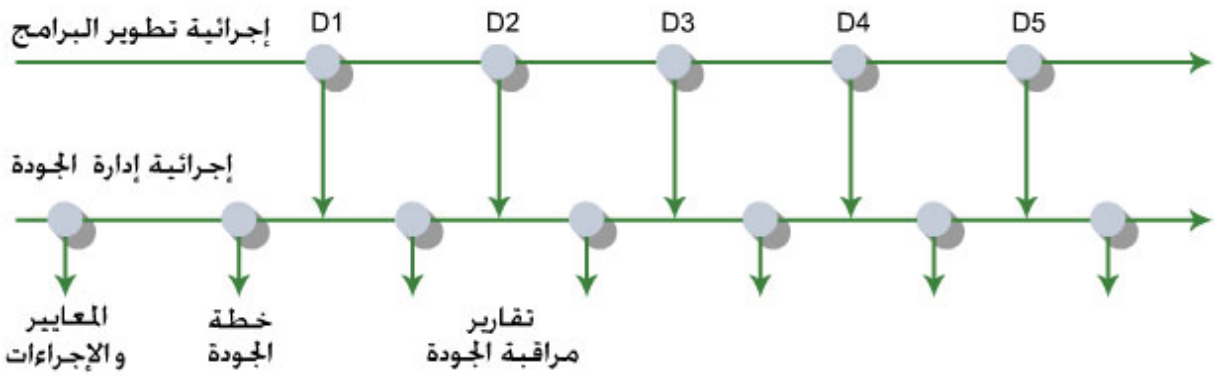
2 - مجالات إدارة الجودة ونشاطاتها

تبرز أهمية إدارة الجودة في الأنظمة الضخمة والمعقدة حيث تسجل وثائق الجودة تقدم التطوير وتدعم استمراره مع تغير فريق التطوير. أما في الأنظمة المتوسطة والصغيرة فهنالك حاجة أقل لهذه الوثائق بينما يكفي التركيز على بناء ثقافة الجودة في المؤسسة.

أهم نشاطات إدارة الجودة:

1. ضمان الجودة: وضع معايير وإجراءات مؤسساتية للجودة.
2. التخطيط للجودة: اختيار المعايير والإجراءات القابلة للتطبيق في مشروع ما وتعديلها حسب الحاجة.
3. مراقبة الجودة: ضمان إتباع معايير وإجراءات الجودة من قبل فريق التطوير.

يجب أن يكون فريق إدارة الجودة مستقل عن فريق إدارة المشروع حتى يحكم بموضوعية على البرمجيات ويراقب التطوير وينقل المصاعب والمشاكل إلى إدارة المؤسسة، كما يجب أن تجري إجرائيات إدارة الجودة على التوازي مع إجرائيات التطوير.



2. جودة المنتجات والإجرائيات

تتأثر جودة المنتج المطور بجودة إجرائية التطوير. يكون هذا الارتباط واضحاً في التصنيع حيث تجري عمليات إعداد الآلات وتشكيلها وضبطها ومعايرتها لتنتج منتجات عالية الجودة . أما في البرمجيات فالعلاقة بين جودة المنتج وإجرائيات التطوير تكون أكثر تعقيداً إذ أن البرمجيات لا تُصنع بل تُصمم، وتطويرها هو عمل إبداعي غير آلي يتأثر بالمهارات الفردية والخبرات وعوامل خارجية أخرى كحداثة التطبيق والضغط التجاري لإصدار مبكر. فمن الصعب مثلاً قياس واصفات جودة مثل قابلية الصيانة. مع ذلك فإن إدارة جودة الإجرائيات يمكن أن تؤدي إلى منتجات برمجية بعيوب أقل.

تتضمن إدارة جودة الإجرائيات النشاطات العملية التالية:

1- تعريف معايير الإجرائيات مثل كيفية إجراء المراجعات وإدارة التشكيلات.

2- مراقبة إجرائية التطوير لضمان إتباع المعايير.

3- إعطاء تقارير عن الإجرائية لإدارة المشروع وإدارة المؤسسة.

يجب الانتباه إلى أن معايير الإجرائيات يجب أن تأخذ بعين الاعتبار نوعية النظام. ففي الأنظمة الحرجة مثلاً يمكن أن تفرض معايير الإجرائيات ضرورة اكتمال التوصيف وإقراره قبل البدء بالتنجيز إلا أن بعض الأنظمة قد يتطلب بناء نموذج مخبري قبل اكتمال التوصيف.

3. ضمان الجودة ومعاييرها

ضمان الجودة هو الإجرائية التي تحدد كيفية الوصول إلى جودة البرمجيات وكيف تعرف المؤسسة أن البرمجيات تملك المستوى المطلوب من الجودة. تُعتبر المعايير مفتاحاً لإدارة الجودة الفعالة. يمكن أن تكون المعايير من المستوى العالمي أو الوطني أو على مستوى المؤسسة أو المشروع. يمكن تعريف نوعين من المعايير كجزء من إجرائيات ضمان الجودة:

1- **معايير المنتج**: التي تعرف ميزات يجب أن تحترمها جميع المكونات مثل أسلوب البرمجة. تتضمن معايير الوثائق والتوثيق والبرمجة.

2- **معايير الإجرائية**: التي تعرف كيفية سير الإجرائية البرمجية، يمكن أن تتضمن تعاريف التوصيف والتصميم والإقرار والوثائق التي ستصدر في كل مرحلة.

هنالك ارتباط وثيق بين معايير المنتج ومعايير الإجرائية، فمعايير المنتج تُطبق على مخرجات الإجرائيات البرمجية ومعايير الإجرائية تتضمن نشاطات تضمن تطبيق معايير المنتج.

1- أهمية المعايير في التطوير البرمجي

المعايير البرمجية مهمة لعدة أسباب:

1- تختزل المعايير معرفة المؤسسة وخبرتها وأفضل الممارسات التي قامت بها مما يمنع من تكرار الأخطاء السابقة.

2- تُعتبر إطار عمل لإجرائيات ضمان الجودة التي تتأكد من وضع معايير مناسبة واستخدامها.

3- توفر المعايير الاستمرارية فيمكن للعناصر الجديدة في الفريق فهم المؤسسة من خلال فهم المعايير المستخدمة.

إن بناء معايير مشاريع هندسة البرمجيات هي عملية صعبة وتستهلك وقتاً طويلاً. كان للمؤسسات الوطنية والدولية الكبيرة مثل US DoD, ANSI, BSI, NATO, IEEE مساهمات كبيرة في وضع معايير عامة يمكن الاستفادة منها وتطبيقها على المشاريع. تُغطي هذه المعايير اصطلاحات هندسة البرمجيات ولغات البرمجة مثل JAVA و C++ والتدوينات كرموز المخططات والإجرائيات والمتطلبات وإجرائيات ضمان الجودة وإجرائيات التأكد والإقرار.

تقوم فرق ضمان الجودة بتحديد المعايير وتقوم بكتابة وثيقة للمعايير بدءاً من المعايير الوطنية أو العالمية المناسبة. يمكن أن تتضمن هذه الوثيقة على سبيل المثال لا الحصر المعلومات الموجودة في الجدول 1:

معايير المنتج	معايير الإجرائية
استمارة مراجعة التصميم	كيفية مراجعة التصميم
بنية وثيقة المتطلبات	إرسال الوثائق إلى إدارة الزبون
صيغة ترويسات الطرائق	إجرائية إصدار نسخة
أسلوب برمجة JAVA	إجرائية الموافقة على خطة المشروع
صيغة خطة المشروع	إجرائية إدارة التغيير
استمارة طلب التغيير	إجرائية تسجيل الاختبار

2- مشاكل المعايير

رغم أهمية المعايير وفائدتها لكنها يمكن أن تطرح مجموعة من المشاكل مثل:

1- يمكن لمهندسي البرمجيات أن يعتبروا أن المعايير غير مناسبة أو غير حديثة أو أنها تولد لديهم شعوراً بالبيروقراطية.

2- إذا لم تكن المعايير مدعومة بأدوات برمجية، فهنالك عمل يدوي ممل لضمان توافق الوثائق مع المعايير.

لتجنب هذه المشاكل يجب على مدراء الجودة أن يقوموا بالخطوات التالية:

1- مشاركة المهندسين في وضع المعايير كي يحيطوا بأسباب اعتمادها.

2- مراجعة المعايير وطرق استخدامها دورياً لأن المعايير يمكن أن تصبح قديمة سريعاً مما يقلل من أهميتها لدى الممارسين.

3- يجب أن تترافق المعايير التفصيلية بأدوات برمجية، إذ أن العمل والعبء الإضافي هو الشكوى الأهم عن المعايير.

3- المعيار ISO 9000

وهي مجموعة عالمية من المعايير لإدارة الجودة يمكن تطبيقها على الكثير من المؤسسات من تلك التي تهتم بالتصنيع إلى تلك التي تهتم بتقديم الخدمات. **المعيار ISO 9001** هو المعيار المناسب للمؤسسات التي تقوم بتصميم وتطوير وصيانة المنتجات، وهو الأكثر عمومية من بين هذه المعايير، كما أنه يقدم نموذجاً عاماً لجودة الإجرائيات يمكن تكييفه حسب المؤسسة.

تصف الوثيقة **ISO 9000-3** هذا المعيار من وجهة نظر هندسة البرمجيات انظر الجدول التالي:

مسؤوليات الإدارة	نظام الجودة
مراقبة التصميم	مراقبة المنتجات التي لا تتوافق مع المعايير
معالجة، تخزين، حزم، توصيل	شراء
منتجات موردة	تحديد المنتجات وتتبعها
مراقبة الإجرائية	تفحص واختبار
مراجعة العقد	إجراء تصحيحي
مراقبة التوثيق	تسجيلات الجودة
تدقيق داخلي للجودة	تدريب
خدمات	تقنيات إحصائية

يجب أن توثق معايير الجودة وإجرائياتها في كتيبات خاصة بالمؤسسة لتعريف إجرائية الجودة. يمكن أن تكون هنالك هيئات خاصة بالمواصفات تمنح المؤسسة شهادة جودة تقضي بأن كتيبات الجودة لديها

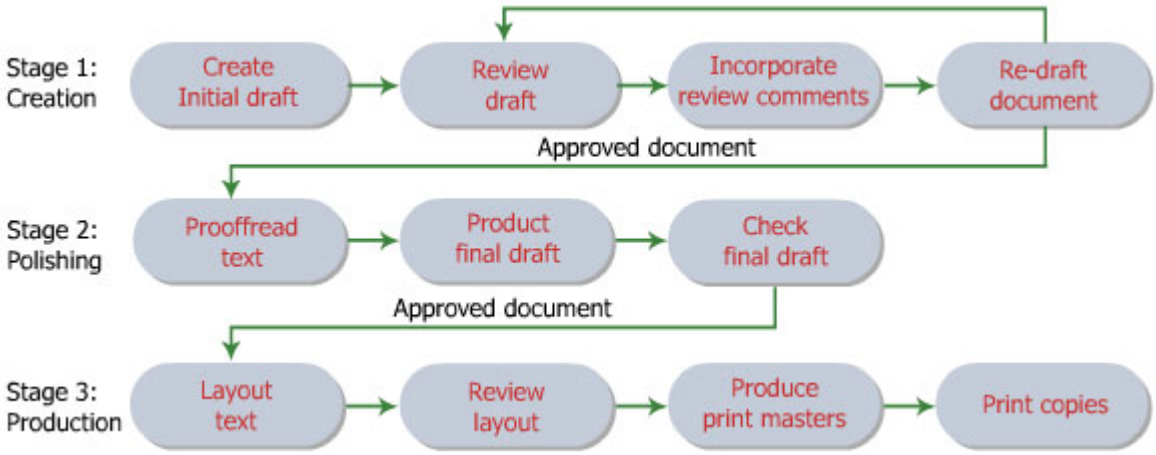
متوافقة مع المعايير ISO 9000 وأن تعريف الإجرائيات المستخدمة والوثائق المرافقة متوافقة مع المعايير، لكن ذلك لا يعني بالضرورة جودة المنتج النهائي.

4- معايير التوثيق

الوثائق هي وسيلة هامة لأنها التعبير الملموس عن البرمجيات وعن الإجرائيات البرمجية. تملك الوثائق المعيارية أشكالاً وبنىً متجانسة مما يسهل قراءتها وفهمها. هنالك ثلاثة أنواع من معايير التوثيق:

معايير إجرائية التوثيق

وتهتم بكيفية إنشاء الوثائق وإقرارها وصيانتها. يجب أن تكون معايير جودة إجرائيات التوثيق مرنة وقابلة للتوافق مع كافة أنواع الوثائق. الوثائق الهامة هي الوثائق الرسمية التي ستستخدم للمراحل التالية من التطوير أو التي ستكون موجهة للزبون. يقدم الشكل التالي نموذجاً لمعايير إجرائية التوثيق.



معايير الوثائق

وتهتم بمحتويات الوثائق وبنيتها وشكلها. ويجب أن تُطبق على جميع الوثائق التي تنتج عن تطوير البرمجيات. يجب أن تكون للوثائق أشكال وبنى متجانسة. يجب أن يكون هنالك معايير خاصة بالمؤسسة ومعايير خاصة بكل مشروع.

يمكن على سبيل المثال وضع المعايير التالية للوثائق:

1- معايير تحديد الوثائق بصورة وحيدة من خلال معايير تسمية مناسبة.

2- معايير بنية الوثائق التي تتعلق بنوع الوثيقة وتعرف أجزاءها وفقراتها وطرق الترقيم وترويسات الصفحات.

3- معايير عرض الوثائق وأنماط الخطوط وأشكالها وقياساتها، استخدام رمز تعريف الشركة، الألوان، والمحددات الأخرى التي تتعلق بالشكل.

4- معايير تغيير الوثائق التي تحدد كيفية عكس تغييرات إصدار معين ضمن وثيقة ما.

معايير تبادل الوثائق

وتتعلق بتوافقية الوثائق الالكترونية وتبادلها عبر البريد الالكتروني والوسائل الأخرى. فالوثائق يمكن أن تكون ناتجة عن أنظمة مختلفة على حواسيب مختلفة. حتى عند استخدام أدوات معيارية محددة، تبقى المعايير ضرورية لتعريف الاصطلاحات واستخداماتها.

4. التخطيط للجودة

توضح خطة الجودة مستوى الجودة المطلوب للمنتجات وكيفية الوصول إلى ذلك المستوى كما تعرف الواصفات الأكثر أهمية للجودة. يجب أيضاً أن تعرف خطة الجودة إجرائية تأمين الجودة والمعايير المؤسساتية التي يجب تطبيقها، وأن تضع معايير جديدة عند الضرورة.

1- بنية خطة الجودة

يجب أن تكون خطط الجودة مختصرة وموجزة لكن بليغة وشاملة، يمكن مثلاً أن تتضمن الفقرات التالية:

1- **تقديم المنتجات**: وصف المنتجات والسوق الموجهة إليه وتوقعات الجودة.

2- **خطة الإنتاج**: التواريخ الهامة والمسؤوليات مع خطط التوزيع والخدمات.

3- **وصف الإجرائيات**: إجرائيات التطوير والخدمات التي يجب استخدامها لتطوير المنتج وإدارته.

4- **أهداف الجودة**: أهداف وخطط الجودة بما فيها تعريف المنتج بصورة وحيدة وشرح أهمية واصفات الجودة.

5- **المخاطر وإدارتها**: المخاطر التي يمكن أن تؤثر على جودة المنتج وكيفية مواجهتها.

هنالك عدد كبير من واصفات الجودة المحتملة للبرمجيات (انظر الجدول التالي) التي يمكن اعتبارها عند وضع خطة الجودة، لكن عادةً لا يكون من الممكن تحقيقها جميعاً لذلك يجب تعريف الواصفات الأكثر أهمية.

الأمان	قابلية الفهم	المحمولية
الأمن	قابلية الاختبار	قابلية الاستخدام
الوثوقية	قابلية التكيف	إعادة الاستخدام
المرونة	الاجتزائية	الفعالية
المتانة	التعقيد	سهولة التعلّم

واصفات جودة البرمجيات

5. مراقبة الجودة

تتضمن مراقبة الجودة تفحص إجرائية التطوير البرمجي لضمان إتباع المعايير والإجراءات المحددة وتطبيقها.

هنالك نهجين أساسيين متكاملين يمكن استخدامهما لمراقبة الجودة:

1- مراجعة الجودة وإبلاغ إدارة المشروع عن الانحرافات عن المعايير.

2- استخدام برمجيات تسمح بمقارنة البرامج والوثائق مع المعايير يمكن أن تتضمن طرق قياس مواصفات برمجية معينة.

تُعتبر مراجعة الجودة الطريقة الأساسية لإقرار جودة منتج أو إجرائية برمجية ما. حيث تقوم مجموعة من الأشخاص بتفحص البرمجيات والوثائق للبحث عن أخطاء أو ملاحظات أو مشاكل محتملة. هنالك عدة أنواع من المراجعات لكل منها أهداف مختلفة:

1- تفتيش عن العيوب بهدف التخلص منها (**جودة المنتجات**): تفحص البرامج والتصاميم للكشف عن الأخطاء في المتطلبات أو التصميم أو الرماز باستخدام قائمة تفحص تحتوي الأخطاء المحتملة.

2- مراجعات تضمن تقدم العمل في المشروع (**جودة المنتجات والإجرائيات**) حسب الخطط الموضوعة والتكاليف والجداول الزمنية مع إعلام الإدارة عن الانزياحات.

3- مراجعات الجودة (**المنتجات والمعايير**): تحليل تقني لمكونات المنتج ووثائقه لكشف عدم التجانس بين المتطلبات والتصميم والرماز والوثائق وخطط الاختبار، ولضمان تطبيق معايير الجودة والمصادقة على البرمجيات والوثائق.

يجب أن تكون فرق مراجعة الجودة صغيرة نسبياً (3 أشخاص مثلاً)، كما يجب أن تكون جلسات المراجعة قصيرة وأن تنتهي بوضع سجل توثيقي لنتائج المراجعة.

يمكن تصنيف الملاحظات الناتجة عن المراجعة إلى الأنواع التالية:

1- ملاحظات لا تؤدي إلى أي عمل مطلوب أو تغيير على البرمجيات أو الوثائق.

2- أخطاء في المتطلبات يجب إعادتها إلى الزبون.

3- ملاحظات مرجعية للتصحيح يجب أن يقوم المصممون أو المبرمجون بتصحيحها.

4- ملاحظات تستوجب إعادة كاملة للتصميم لأن المشاكل المكتشفة يمكن أن تؤثر على أجزاء أخرى. يجب أحياناً التفكير في الطريقة الأجدى من حيث التكلفة لحل المشكلة.

6. قياس البرمجيات ومقاييس البرامج

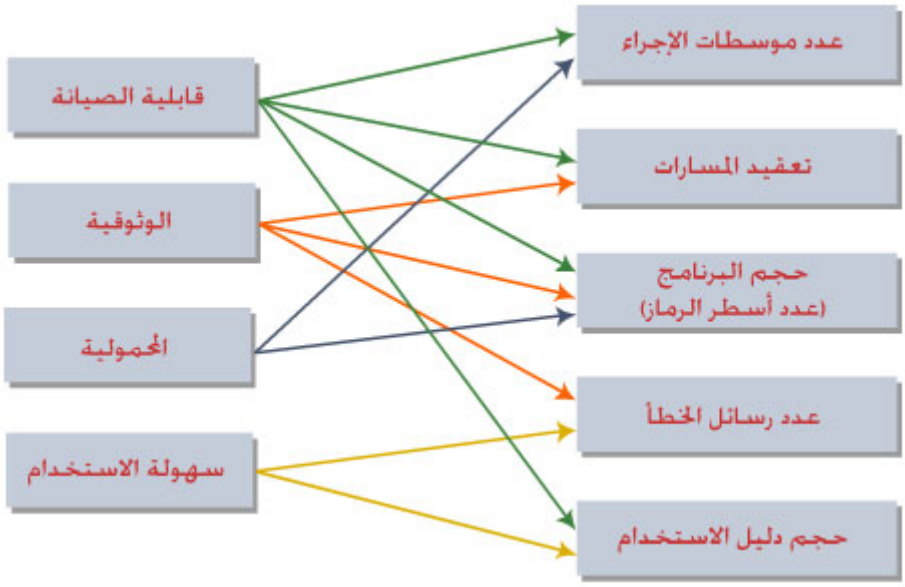
يهتم قياس البرمجيات بتحديد قيمة رقمية لكل واصفة من واصفات المنتج أو الإجرائية. يسمح ذلك بمقارنة موضوعية. أنتجت بعض الشركات برامجاً خاصة لقياس البرمجيات تسمح بتقييم مواصفات

البرمجيات. مع ذلك فإن معظم المؤسسات مازالت لا تعتبر أن قياس البرمجيات هو إجراء أساسي وهام. كما أن المعايير التي تستخدم لقياس البرمجيات قليلة.

مقاييس البرمجيات هي جميع أنواع القياسات المتعلقة بالأنظمة البرمجية أو الإجرائيات أو الوثائق مثل عدد أسطر الرماز في برنامج ما، أو الجهد المطلوب لإنجاز إحدى المكونات (مقدراً برجل / شهر).

تسمح هذه المقاييس بتحديد قيم للبرمجيات والإجرائيات البرمجية، ويمكن أن تُستخدم لتوقع واصفات المنتج أو التحكم بالإجرائية البرمجية، أو تحديد المشاكل في المكونات، مما يمكن ويسهل اتخاذ قرارات بشأنها من قبل الإدارة.

من غير الممكن إجراء قياس مباشر لواصفات جودة البرمجيات التي تُعتبر مواصفات خارجية يراها المستخدم أو المطور كقابلية الصيانة وسهولة الاستخدام. فهذه الواصفات تتأثر بعدة عوامل ولا يمكن قياسها بطريقة بسيطة. في المقابل يمكن قياس بعض الواصفات الداخلية للبرمجيات مثل الحجم، ثم افتراض وجود علاقة بين ما يمكن قياسه وما يجب معرفته. في الحالة المثالية تكون هذه العلاقة واضحة ومقررة. يظهر الشكل التالي بعض واصفات الجودة الخارجية والواصفات الداخلية التي يمكن أن ترتبط بها دون إظهار طبيعة الارتباط.



1- إجرائية القياس

يمكن أن تكون إجرائية القياس جزءاً من إجرائية مراقبة الجودة. يجب أن يجري الاحتفاظ بالمعطيات التي جمعت خلال هذه الإجرائية كمورد في المؤسسة. عند اعتماد قاعدة معطيات للقياسات يصبح من السهل مقارنة المشاريع المختلفة.

تمر إجرائية القياس بالمراحل التالية:

1- اختيار القياسات التي يجب إجراؤها.

2- اختيار المكونات التي سيجري قياسها.

3- قياس مواصفات المكونات.

4- تحديد القياسات الشاذة.

5- تحليل المكونات الشاذة.

يجب أن يعتمد برنامج قياس البرمجيات على مجموعة من المعطيات الخاصة بالمنتج والإجرائية. يجب أن تُجمع المعطيات آنياً عند صدورها وليس في نهاية المشروع، وإذا أمكن تجميعها آلياً. هنالك ثلاثة أنواع من التجميع الآلي للمعطيات:

1- التحليل الساكن للمنتج.

2- التحليل الديناميكي للمنتج.

3- جميع معطيات الإجرائية.

من الضروري التشديد على صحة المعلومات المجمعة من خلال تحديد الأسئلة والمعطيات المطلوبة سلفاً. كما يجب إعلام الأشخاص بالهدف من تجميع المعطيات وأن لا يكون ذلك جزءاً من تقييم الأشخاص.

2- مقاييس المنتج

يجب أن يكون مقياس جودة المنتجات وسيلة لتوقع جودتها. هنالك نوعين من مقاييس المنتج:

1- المقاييس الديناميكية التي تُجمع من خلال قياسات تُجرى أثناء تنفيذ البرنامج: تساعد هذه المقاييس في تحديد الفعالية والوثوقية. وهي ترتبط بشدة بواصفات جودة البرمجيات. وتكون سهلة القياس نسبياً (زمن استجابة النظام، عدد الأعطال).

2- المقاييس الساكنة التي تُجمع من خلال قياسات تُجرى على نماذج النظام: تساعد هذه المقاييس في تحديد التعقيد وسهولة الفهم والصيانة. وتكون علاقتها بواصفات الجودة غير مباشرة . ويكون من الضروري تحديد العلاقة بينها وبين واصفات الجودة كالتعقيد وقابلية الصيانة وسهولة الفهم والاستخدام.

يظهر الجدول التالي عدة قياسات ساكنة للمنتج تُستخدم لقياس الجودة:

قياس البرمجيات	الوصف
الدخلية والخرجية fan-in/fan-out	تمثل الدخلية عدد التوابع أو الطرائق التي تستدعي تابعاً أو طريقة معينة. بينما تمثل الخرجية عدد التوابع أو الطرائق التي يستدعيها تابع أو طريقة معينة. إنَ ارتفاع قيمة الدخلية يعني أن التصميم مرتبط بشدة بالطريقة المعنية كما يدل على شدة إعادة الاستخدام. بينما يدل ارتفاع قيمة الخرجية على زيادة تعقيد الطريقة المعنية

	وتعقيد التحكم فيها.
عدد أسطر الرماز	يدل على تعقيد البرنامج واحتمال الخطأ فيه.
تعقيد المسارات	قياس لدرجة تعقيد التحكم التي تحدد سهولة الفهم.
طول أسماء المتحولات	يدل طول هذه الأسماء على كونها معبرة وتساعد في فهم البرنامج.
عمق التداخل الشرطي	التداخل الكبير في التعليمات الشرطية IF يدل على صعوبة الفهم واحتمال أكبر للخطأ.
مؤشر fog	قياس لطول الكلمات والجمل في الوثائق تدل زيادته على صعوبة فهمها.

القياسات السابقة عمومية يمكن تخصيصها أكثر، ويمكن أيضاً أن نستعرض قياسات أخرى غرضية التوجه كما في الجدول التالي:

القياس الغرضي التوجه	الوصف
عمق شجرة الوراثة	يظهر ذلك في مخطط الصفوف وتدل الشجرة العميقة على صعوبة الفهم وتعقيد التصميم.
تثقيل الطرائق في كل صف	عدد الطرائق في كل صف مع تثقيلها حسب تعقيدها. يحدد هذا القياس درجة التعقيد ومن ثم سهولة الفهم.
عدد العمليات المبطلة في الهرمية	Overriding operations. يدل ارتفاع هذا العدد على أن الصف الأب لصف ما قد لا يكون مناسباً له.

3- تحليل القياسات

إن تحليل المعطيات المجمعة هو المرحلة الأكثر صعوبةً في عملية القياس فليس من البديهي دوماً فهم معنى المعطيات المجمعة وفائدتها، يجب الاستعانة بخبرة استشاريين إحصائيين مختصين يعملون بأسلوب مهني، ويجب أخذ الظروف المحلية للمشروع والواقع الراهن للمؤسسة بعين الاعتبار.

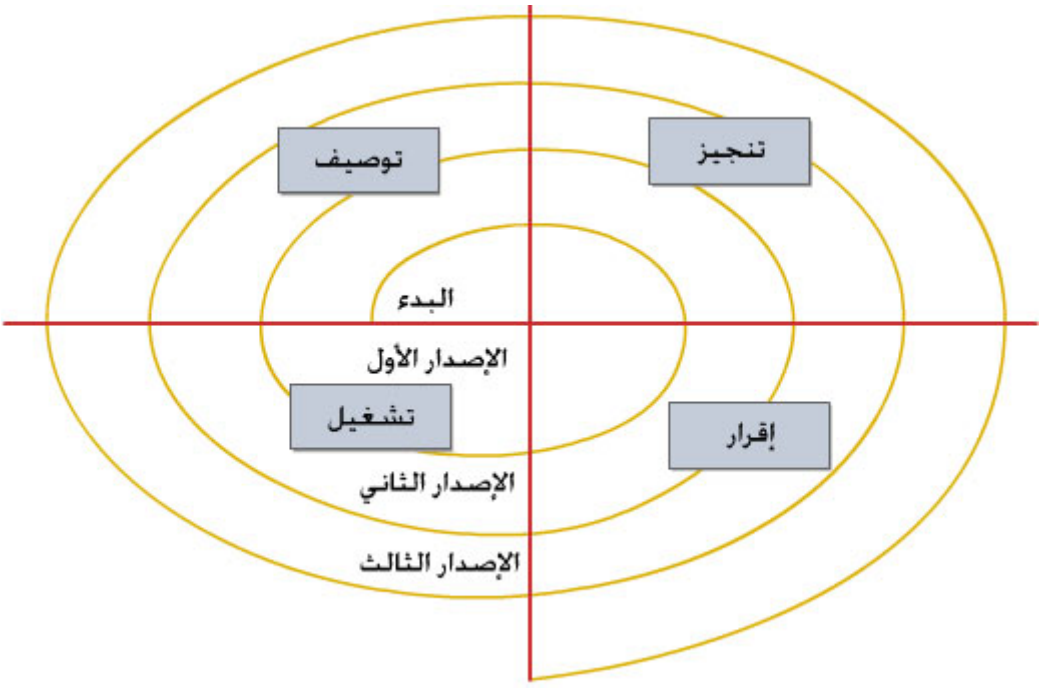
لا يخلو تحليل القياسات من المفاجآت لكن يجب تحليل هذه المفاجآت ومعرفة أسبابها، من ذلك مثلاً أن تقليل عدد الأخطاء في البرنامج يؤدي إلى ارتفاع عدد الطلبات على منصة المساعدة. تفسير ذلك قد يكون بأن البرنامج أصبح أكثر وثوقية ومن ثم فقد أصبح له سوق أوسع وأكثر تشعباً. ويعني ذلك أن نسبة المستخدمين الذين يطلبون منصة المساعدة إلى العدد الكلي للمستخدمين قد قلّت، لكن إجمالي عددهم قد ازداد لازدياد عدد مستخدمي النظام. تفسير آخر يمكن أن يكون منطقياً هو أن النظام الأكثر وثوقية يستخدم بطريقة مختلفة من قبل المستخدمين لا تستدعي الالتفاف حول الأخطاء مما يعني طلبات أكثر لمنصة المساعدة. هذا النوع من التحليلات ينتج عن الخبرة الطويلة في تطوير البرمجيات وهو بالغ الأهمية عند مراجعة نتائج القياسات.

1. التغيير في البرمجيات

إن من المشاكل الهامة التي تواجه المؤسسات هي إدارة التغيير وتنجيز تغييرات على الأنظمة البرمجية الخاصة بها. لكن من غير الممكن تجنب تغيير وتطور البرمجيات خلال استخدامها وإلا فإنها ستكون مفيدة لمرحلة مؤقتة ثم تصبح غير متناسبة مع الأعمال. يمكن أن نذكر عدة أسباب للتغيير في البرمجيات منها:

1- ظهور متطلبات جديدة في الأعمال.
2- تغير بيئة العمل.
3- ظهور أخطاء تحتاج إلى تصحيح.
4- إضافة حواسيب وتجهيزات جديدة إلى النظام.
5- ضرورة تحسين أداء النظام أو وثوقيته.

كما أن أهمية التغيير والتعديل في الأنظمة البرمجية تأتي من أن هذه الأنظمة تُعتبر من الأصول الهامة في أرصدة الأعمال ومن أن المؤسسات تضع استثمارات ضخمة فيها. لذلك ليس من المستغرب أن نجد أن معظم الميزانية المخصصة للبرمجيات في الشركات الكبيرة تخصص لترقية وتحسين البرمجيات الموجودة بدلاً من تطوير برمجيات جديدة. يمكننا بالنتيجة التفكير بهندسة البرمجيات كإجرائية حلزونية تمتد على مدى حياة النظام (انظر الشكل).



2. ديناميكية ارتقاء البرامج

ديناميكية ارتقاء البرامج هي دراسة تغييرات الأنظمة. بعد دراسات تجريبية هامة قام بها كل من Lehman و Belady اقترحا مجموعة من القوانين التي تنطبق على جميع الأنظمة عند تغييرها وترقيتها (انظر الجدول: قوانين Lehman).

القانون	الوصف
تغيير مستمر	إن برنامجاً مستخدماً في بيئة حقيقية يجب أن يتغير أو أنه سيصبح مع الزمن أقل فائدةً في تلك البيئة.
تعقيد متزايد	عندما يتغير برنامج ما ويرتقي تصبح بنيته أكثر تعقيداً. يجب تخصيص موارد إضافية للحفاظ على البنية وتبسيطها.
ارتقاء كبير للبرامج	ارتقاء البرامج هي إجرائية ذاتية التصحيح. فمواصفات النظام كالحجم والفترة الزمنية بين الإصدارات وعدد الأخطاء هي تقريباً ثابتة في كل إصدار.
استقرار مؤسساتي	نسبة التطوير ثابتة تقريباً على مدى حياة نظام ما ومستقلة عن الموارد المخصصة لذلك.
انحفاظ الألفة	التغيير المتزايد في كل إصدار ثابت تقريباً على مدى حياة النظام.
تضخم مستمر	الوظائف المقدمة من قبل نظام ما يجب أن تزيد باستمرار للحفاظ على رضا المستخدمين.
جودة متناقصة	ستبدو الجودة متناقصة إلا إذا سايرت الأنظمة التغيير في بيئة التشغيل.
نظام تغذية راجعة	تتضمن إجرائيات الارتقاء أنظمة تغذية راجعة يجب الاهتمام بها كمؤشرات هامة عن تحسين المنتج.

يمكن اعتبار هذه القوانين ملاحظات دقيقة تُطبق غالباً على الأنظمة الكبيرة في المؤسسات الضخمة. وليس من الواضح كيفية ملاءمتها للبرامج الأخرى الصغيرة أو المتوسطة أو التي تتضمن مكونات جاهزة أو التي تُنفذ في مؤسسات صغيرة.

3. صيانة البرمجيات

صيانة البرمجيات هي الإجرائية العامة لتغيير البرمجيات والتعديل فيها بعد وضعها في بيئة العمل والاستخدام. لا تتضمن الصيانة عادةً تغييرات كبيرة في بنية النظام بل تكون تغييرات في بعض المكونات الموجودة أو إضافة مكونات جديدة.

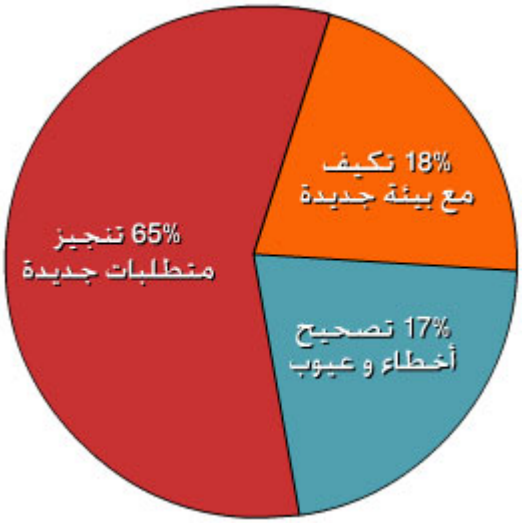
لا يمكن تجنب أعمال الصيانة للبرمجيات فمتطلبات النظام تتغير أثناء استخدامه، وبيئة العمل في تغير دائم مما يجعل استمرار الاستفادة من النظام مرهوناً بصيانته المستمرة.

هنالك ثلاثة أنواع من الصيانة البرمجية:

1- تصحيح أخطاء برمجية أو إصلاح عيوب في النظام أو ملاقاة متطلبات غير متوفرة.
2- تكييف النظام مع بيئة تشغيل تختلف عن بيئة تشغيله الأولى (حواسيب، نظم تشغيل، ...).
3- تعديل في وظائف النظام أو إضافة وظائف جديدة له تعكس متطلبات جديدة.

3. صيانة البرمجيات

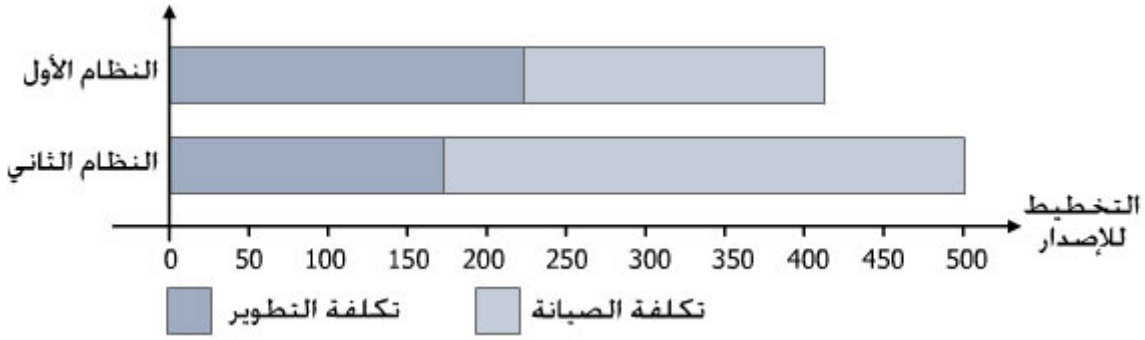
أظهرت مجموعة من الدراسات الإحصائية أن حوالي 65% من الصيانة تتعلق بتنجيز متطلبات جديدة بينما 18% من الصيانة يكون للتكيف مع بيئة جديدة و 17% لتصحيح أخطاء وعيوب.



تكون تكلفة صيانة البرمجيات عادةً أكبر بكثير من تكلفة تطوير البرمجيات وقد تكون ضعفي هذه التكلفة وقد تصل إلى مائة ضعف حسب نوع التطبيق. تتزايد هذه التكلفة باستمرار الصيانة **خاصةً** إذا كانت تلك الصيانة تؤثر على بنية النظام مما يجعل صيانته المستقبلية أكثر صعوبةً. كذلك فإن الأنظمة القديمة المعمرة تصبح تكلفة صيانتها كبيرة مثل الأنظمة التي تستخدم تجهيزات قديمة أو المكتوبة بلغات برمجة قديمة.

3. صيانة البرمجيات

من المجدي دوماً استثمار جهد أكبر في التصميم والتنجيز للخروج بنظام واضح وسهل الفهم والتغيير مما يساهم في تقليل تكاليف الصيانة. يظهر الشكل التالي كيف أن التكلفة العامة للنظام على مدى حياته يمكن أن تصبح أقل عند صرف جهد أكبر خلال التطوير للحصول على نظام سهل الصيانة.



3. صيانة البرمجيات

هنالك مجموعة من العوامل التي تؤثر على تكلفة الصيانة منها:

1- **استقرار الفريق**: تتأثر تكاليف الصيانة كثيراً باستمرار عمل نفس الفريق فيها، فالعناصر الجديدة ستتطلب جهداً ووقتاً كبيراً لفهم النظام وتنجيز تغييرات عليه.

2- **المسؤولية التعاقدية**: قد لا تكون الجهة المطورة هي نفسها المسؤولة عن الصيانة مما يجعل لديها أولويات أخرى في توفير الجهود أثناء التطوير على حساب تكلفة عالية للصيانة.

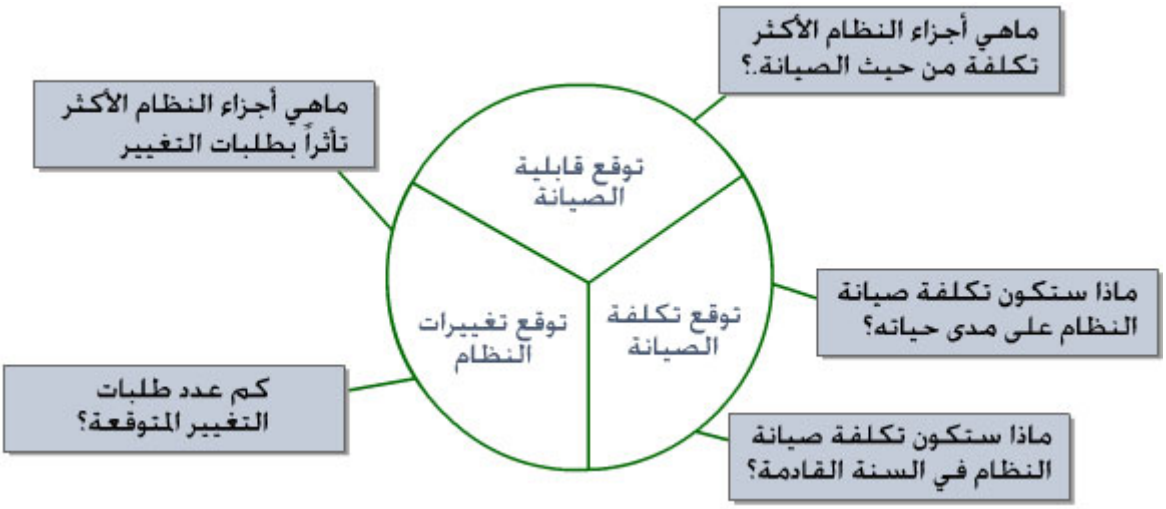
3- **مهارات فريق الصيانة**: يكون فريق الصيانة غالباً قليل الخبرة ولديه معرفة محدودة بمجال العمل.

4- **تعمير البرامج وبنيتها**: مع تقدم البرامج في العمر تصبح بنيتها متشعبة فتصبح صعبة الفهم والصيانة.

3. صيانة البرمجيات

1- توقع الصيانة

يكره المدراء عادةً المفاجآت خاصةً تلك المكلفة منها. يجب إذاً توقع التغييرات على النظام وتحديد الأجزاء التي ستكون صعبة الصيانة فيه. كما يجب تقدير تكاليف صيانة النظام في فترة زمنية معينة. يظهر الشكل التالي هذه التوقعات مع الأسئلة المرافقة لها.



هذه التوقعات والتقديرات مترابطة بشدة:

1- يعتمد قبول التغيير على قابلية صيانة المكونات التي تتأثر بهذا التغيير.

2- تنجيز التغييرات يشتت النظام ويقلل من قابليته للصيانة.

3- تعتمد تكلفة الصيانة على عدد التغييرات وتعتمد تكلفة التغيير على قابلية الصيانة.

2- توقع التغيير

يتطلب توقع عدد التغييرات فهماً عميقاً للعلاقة بين النظام والبيئة التي يعمل فيها فالأنظمة المرتبطة بشدة بالبيئة تتطلب تغييراً بتغيرها. من العوامل التي تؤثر على هذه العلاقة:

1- عدد واجهات النظام ودرجة تعقيدها.

2- عدد متطلبات النظام ذات الطبيعة المتغيرة.

3- عدد إجرائيات الأعمال التي تستخدم النظام.

لتوقع التغيير والصيانة يجب فهم عدد ونوعية العلاقات بين مكونات النظام ودرجة تعقيدها. أثبتت الدراسات الإحصائية أن معظم جهود الصيانة تتركز على عدد قليل من مكونات النظام. يمكن اعتماد مجموعة من القياسات للتعقيد مثل:

1- تعقيد بنى التحكم.

2- تعقيد بنى المعطيات.

3- حجوم الأغراض والإجراءات والطرائق والمجتزءات.

بعد وضع النظام في الاستخدام يمكن استخدام معطيات الإجرائيات لتوقع الصيانة، فيما يلي بعض الأمثلة عن المقاييس التي يمكن أن تُستخدم لذلك:

1- عدد طلبات الصيانة التصحيحية.

2- الزمن الوسطي اللازم لتحليل تأثيرات الصيانة.

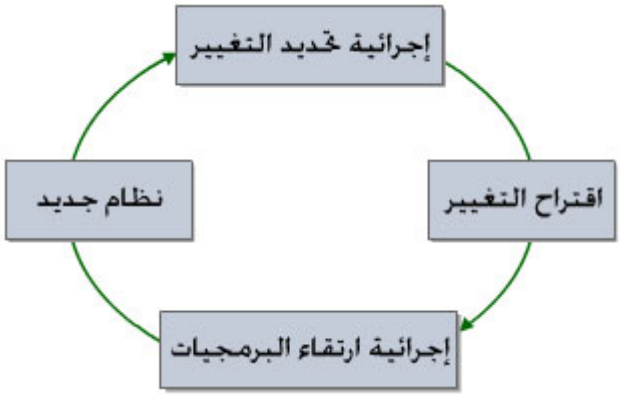
3- الزمن الوسطي اللازم لتنجيز طلب تغيير.

4- عدد طلبات التغيير المنتظرة.

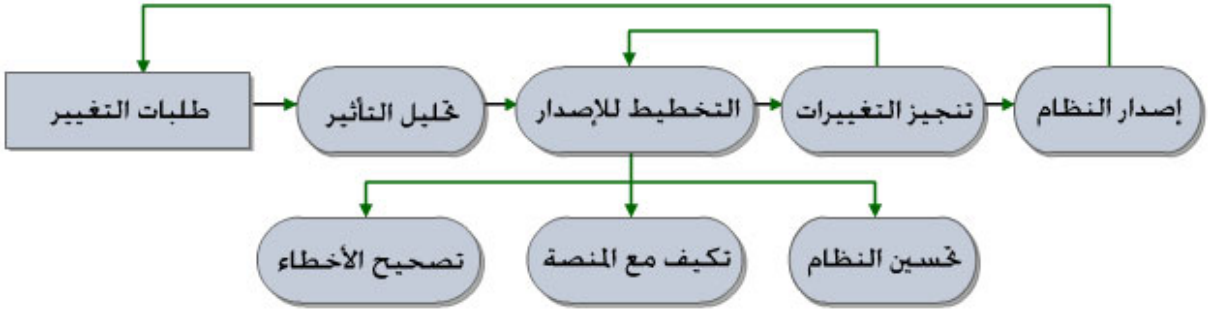
إن تزايد أي من القياسات السابقة يدل على تناقص في قابلية الصيانة.

4. إجرائيات الارتقاء

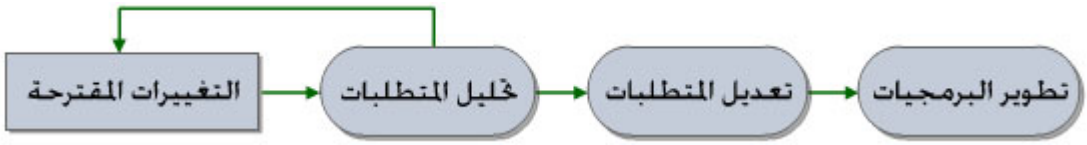
تعتمد إجرائيات الارتقاء على نوعية البرمجيات التي تجري صيانتها، وعلى إجرائية التطوير المستخدمة، ومهارات فريق العمل وخبراته. في بعض المؤسسات يمكن أن يكون التطور غير رسمياً حيث تأتي التغييرات نتيجة مناقشات بين مستخدمي النظام ومطوريه. في مؤسسات أخرى قد تكون له صفة رسمية موثقة. في جميع الأحوال يكون المحرك الدافع للارتقاء هو اقتراحات التغيير. تبدأ العملية من تحديد التغييرات ثم اقتراحها وتستمر خلال فترة حياة النظام (انظر الشكل).



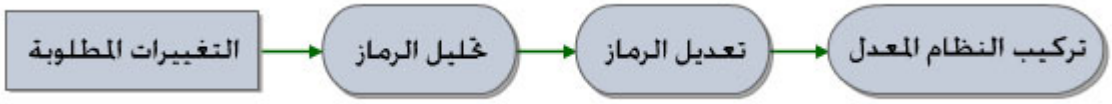
تتضمن إجرائية الارتقاء مجموعة من النشاطات الأساسية أهمها تحليل التغيير والتخطيط للإصدار الجديد وتنجيزه. يظهر الشكل التالي مراحل ونشاطات إجرائية ارتقاء نظام برمجي.



في الحالة المثالية تؤدي مرحلة تغيير التنجيز من هذه الإجرائية إلى تغيير التوصيف والتصميم بما يتوافق مع التغييرات الجديدة (انظر الشكل).



يمكن أن تتعلق طلبات التغيير بمشاكل في النظام يجب معالجتها بسرعة، دون المرور بمراحل إجرائية هندسة البرمجيات حيث يصبح مخطط العمل كما في الشكل التالي.



يمكن أن تظهر التغييرات الإسعافية هذه لثلاثة أسباب:

1- خطأ حقيقي هام في النظام يجب تصحيحه.
2- تغييرات في بيئة النظام لها تأثيرات غير متوقعة (مثل ترقية نظام التشغيل).
3- تغييرات في الأعمال تتطلب استجابة سريعة (كإصدار منتج منافس مثلاً).

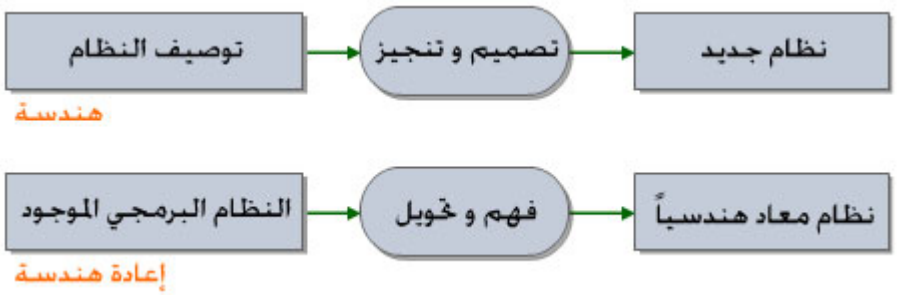
5. إعادة هندسة الأنظمة

إن الارتقاء بنظام ما يتطلب فهم البرنامج الذي يجب تغييره قبل إنجاز التغيير. غالباً ما تكون الأنظمة القديمة صعبة الفهم لكونها بنيت على أساس زيادة الأداء وتقليل حجوم التخزين. لتبسيط عملية تغيير الأنظمة القديمة تلجأ بعض الشركات إلى إعادة هندسة النظام، أي إعادة بناءه أو إعادة كتابة أجزاء منه دون تغيير وظائفه، ليصبح بعد ذلك قابلاً للصيانة. يمكن أن تتضمن إعادة الهندسة إعادة التنجيز والتوثيق أو إعادة التنظيم والبناء أو كتابة البرمجيات بلغة حديثة.

تُطبق إعادة الهندسة عندما تكون بعض أجزاء منظومة برمجية كبيرة تتطلب صيانة مستمرة ومتكررة. حيث يصبح من المفيد إعادة كتابة هذه الأجزاء لزيادة قابلية صيانتها. تتميز إعادة الهندسة عن الطرائق الأخرى للارتقاء بالبرمجيات بفائدتين رئيسيتين:

1- **مخاطرة أقل**: يرتبط تطوير البرمجيات الجديدة دوماً بمخاطر كبيرة تتعلق بالتطوير وفريق العمل ومشاكل التوصيف وعوامل أخرى كثيرة، بينما تكون إعادة الهندسة أقل مخاطرةً.
2- **تكلفة أقل**: تكلفة إعادة الهندسة تكون غالباً أقل بكثير من تكلفة تطوير برمجيات جديدة.

الفرق الهام بين إعادة هندسة نظام ما وتطوير نظام جديد هو نقطة البداية. فبدلاً من البدء من توصيفات مكتوبة يجري عند إعادة الهندسة البدء من نظام برمجي موجود (انظر الشكل).



يظهر الشكل التالي النشاطات التي تتضمنها إجرائية إعادة الهندسة:

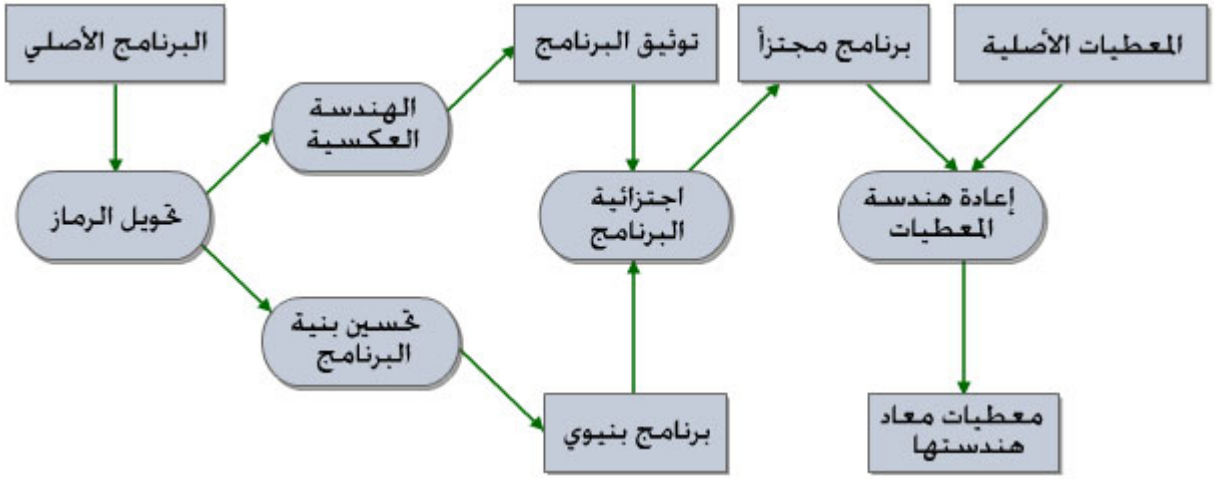
1- **تحويل الرماز المصدر**: تحويل الرماز وكتابته بلغة جديدة أو نسخة جديدة من اللغة الأصلية.

2- **الهندسة العكسية**: تحليل النظام بهدف فهم بنيته ووظائفه.

3- **تحسين بنية البرامج**: تحليل بنية التحكم وجعلها سهلة الفهم.

4- **اجتزائية البرامج**: إعادة تنظيم بنية البرامج بتجميع الأجزاء عند مناسبة ذلك وبإلغاء التكرار عند وجوده. ويمكن أن يتضمن ذلك التحويل من بنية مركزية إلى بنية موزعة.

5- **إعادة هندسة المعطيات**: تنظيف وإعادة تنظيم بنية المعطيات.



تعتمد تكلفة إعادة الهندسة على الجهد المبذول لذلك. فهنالك طيف واسع من الطرق المستخدمة لذلك. تتغير الكلفة بتغير هذه الطرق (انظر محور التكلفة في الشكل التالي).

هنالك عدة عوامل مؤثرة في تكلفة إعادة هندسة البرمجيات:

1- جودة البرمجيات التي سيجري إعادة هندستها.
2- الأدوات المتوفرة لدعم عملية إعادة الهندسة.
3- المدى المطلوب لتحويل وتصحيح المعطيات.
4- توفر فريق عمل خبير في إعادة الهندسة.

6. الارتقاء بالأنظمة القديمة

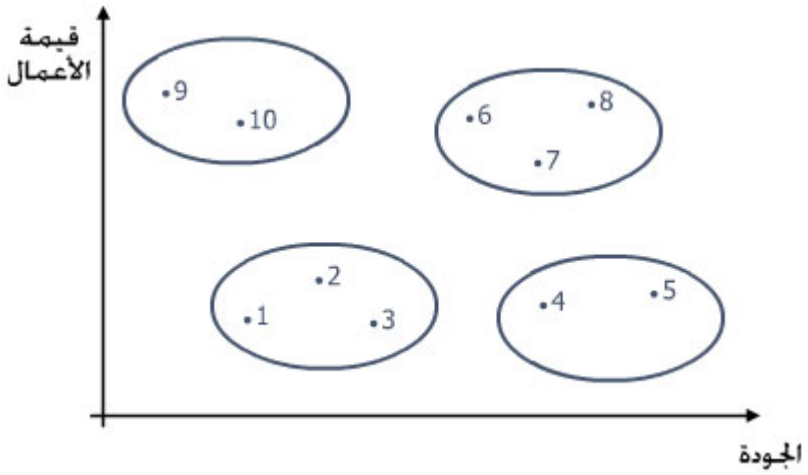
يجب على المؤسسات التي تعتمد على أنظمة قديمة أن تختار استراتيجية مناسبة للارتقاء بهذه الأنظمة وتطويرها من بين الاستراتيجيات التالية:

1- إلغاء النظام نهائياً وتغيير إجرائيات الأعمال بحيث لا تكون هنالك حاجة له.
2- الاستمرار في صيانة النظام.
3- تحويل النظام وإعادة هندسته لتحسين قابليته للصيانة.
4- استبدال النظام بنظام جديد.

تتعلق الاستراتيجية المختارة بجودة النظام وأهميته وقيمته في الأعمال. لتوضيح ذلك سنفترض مؤسسة لديها 10 أنظمة قديمة جرى تحديد جودة وقيمة كل منها ثم جرى تمثيلها في مخطط بالشكل التالي.
نلاحظ من الشكل أن هنالك 4 أصناف من الأنظمة:

1- **جودة منخفضة وقيمة أعمال منخفضة**: هذه الأنظمة يجب إلغاؤها (1، 2، 3).
2- **جودة منخفضة وقيمة أعمال مرتفعة**: مساهمة كبيرة في الأعمال لكن تكاليف صيانة كبيرة. يجب إعادة هندسة هذه الأنظمة أو استبدالها إذا توفر نظام مناسب بديل (9، 10).
3- **جودة عالية وقيمة أعمال منخفضة**: يمكن استبدالها بمنتجات جاهزة إذا توفرت، أو إلغاءها كلياً أو الاستمرار بصيانتها (4، 5).

4- **جودة عالية وقيمة أعمال مرتفعة**: يمكن استمرار التشغيل باستخدام الصيانة الطبيعية للنظام (6، 7، 8).



يجب أن يتناول تحديد قيمة الأعمال لنظام معين عدة وجهات نظر منها:

1- المستخدمون النهائيون للنظام.

2- زبائن المؤسسة.

3- مدراء خطوط التشغيل.

4- مدراء التقانة.

5- المدراء في المستوى الأعلى للمؤسسة.

أما تقييم الجودة فيتعلق بثلاثة نواحي هامة:

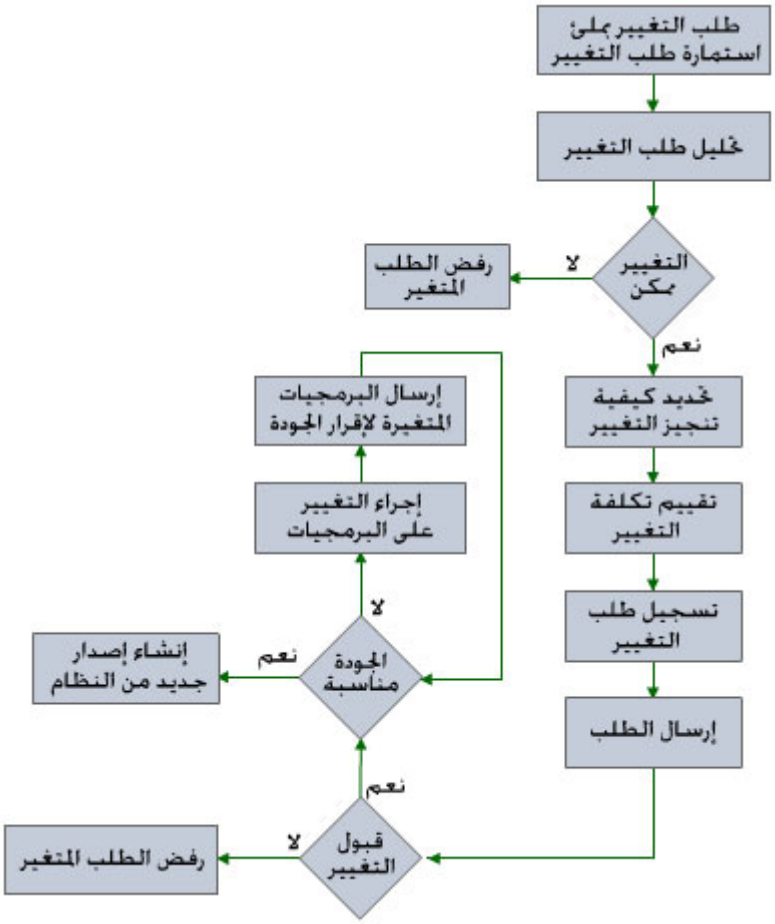
1- **تقييم إجرائيات الأعمال**: إلى أي درجة تدعم إجرائية الأعمال الأهداف الحالية للمؤسسة.

2- **تقييم البيئة**: مدى فعالية البيئة الحالية وما تكاليف صيانتها.

3- **تقييم التطبيق**: ما هي جودة النظام البرمجي للتطبيق.

7. إدارة التغيير

تهتم إدارة التغيير بتسجيل التغييرات التي تجري على النظام بطلب من المستخدمين أو المطورين أو بدافع من تغييرات السوق وضمان تنفيذها بأفضل طريقة من حيث الجدوى والتكلفة.
هنالك إجراءات معروفة ومحددة لإدارة التغيير تبدأ من تحليل التكلفة والفائدة للتغييرات المقترحة ثم تحديد المكونات التي سيجري تغييرها (انظر إجرائية إدارة التغيير في الشكل التالي).



1 طلب التغيير بملئ استمارة طلب تغيير

2 تحليل طلب التغيير

3- إذا كان التغيير ممكناً

3-1- تحديد كيفية تنجيز التغيير

3-2- تقييم تكلفة التغيير

3-3- تسجيل طلب التغيير في قاعدة المعطيات

3 4 إرسال الطلب إلى هيئة إدارة التغيير

3 5 عند قبول التغيير

3-5-1- تكرار

3-5-1-1- إجراء التغيير على البرمجيات

3-5-1-2- إرسال البرمجيات المتغيرة لإقرار الجودة

3-5-2- إلى أن تكون الجودة مناسبة

3 5 3 إنشاء إصدار جديد من النظام

3 6 في حال عدم قبول التغيير

3-6-1- رفض طلب التغيير

4- فيما عدا ذلك

4-1- رفض طلب التغيير

يجري طلب التغيير باستمارة خاصة يملؤها طالب التغيير تتضمن التغيير المقترح وطالب التغيير وأسباب التغيير ومدى العجلة المطلوبة لإجراءه. كما تتضمن تقييم التغيير، وتكلفته والتوصيات من قبل فريق الصيانة.

من المهم جداً تسجيل حالات التغيير. توفر الأدوات المساعدة في هندسة البرمجيات أدوات خاصة لإدارة التغيير، تقوم بتسجيل حالات طلبات التغيير وترسلها إلى الأشخاص المعنيين في الوقت المناسب، يمكن مكاملة هذه الأدوات مع البريد الإلكتروني لتوزيع طلبات التغيير مباشرةً.

يجب مراجعة التغييرات من قبل مجموعة خارجية مستقلة عن إدارة المشروع يمكن أن تضم ممثلين عن الزبائن وأطراف العقد، تُقرر من وجهة نظر استراتيجية مؤسساتية جدوى التغيير مقابل التكلفة.

يكون هنالك عادةً مع كل مكون من مكونات النظام سواء كان برنامج أو وثيقة سجل تاريخي للتغيرات. يسجل في هذا السجل التغييرات المجراة وأسبابها ومن الذي قام بها ومتى. يمكن وضع هذه المعلومات كملاحظات على الرماز (انظر المقطع البرمجي في **الشكل التالي**). كما يمكن الاستفادة منها ومعالجتها آلياً بوساطة أدوات برمجية إذا كانت تتبع أسلوب كتابة ثابت ومحدد.

```
// BANKSEC project (IST 6087)
//
// BANKSEC-TOOLS/AUTH/RBAC/USER_ROLE
//
// Object: currentRole
// Author: N. Perwaiz
// Creation date: 10th November 2002
//
// © Lancaster University 2002
//
// Modification history
// Version      ModifierDate            Change          Reason
// 1.0    J. Jones        1/12/2002       Add header      Submitted to CM
// 1.1    N. Perwaiz      9/4/2003New field       Change req. R07/02
```

8. إدارة التشكيلات والإصدارات والسحوب Configuration, Version and Release management

مع تطور النظام وارتقائه وحصول تغييرات عليه يجري إنشاء إصدارات جديدة منه:

1- لأنواع مختلفة من الحواسيب أو أنظمة التشغيل.

2- لتقديم وظائف جديدة.

3- لملاءمة استخدام معين لزبون معين.

تهتم إدارة التشكيلات بإدارة التطورات التي تطرأ على النظام، فهي عمل جماعي للفريق هدفه التحكم بالتكلفة والجهد المرافق لإجراء التغييرات على النظام. يتضمن ذلك تطوير إجراءات خاصة وتطبيقها وقد يكون ضمن مجال إدارة الجودة. يمكن اعتماد معايير لتحديد كيفية تسمية الإصدارات والتحكم بالتغييرات وإدارة الإصدارات الجديدة.

قد يكون من الضروري إدارة كافة منتجات الإجرائية البرمجية من توصيفات وتصاميم وبرامج ومعطيات اختبار وأدلة استخدام. يمكن أن يكون هنالك آلاف الوثائق المولدة لنظام برمجي كبير ومعقد.

يجب وضع **خطة لإدارة التشكيلات** تتضمن هذه الخطة:

1- أنواع الوثائق التي يجب إدارتها.
2- آلية تسمية للوثائق.
3- تحديد مسؤوليات إجراءات إدارة التشكيلات.
4- تعريف سياسات التغيير وإدارة الإصدارات.
5- تعريف سجلات قاعدة معطيات إدارة التشكيلات التي يجب استخدامها.
6- تحديد الأدوات المساعدة التي يجب استخدامها وطريقة استخدامها.

للأدوات المساعدة في هندسة البرمجيات CASE tools أهمية خاصة في إدارة التشكيلات. فيمكن من خلالها تطبيق معايير وإجراءات معرفة سلفاً. تتوفر الكثير من الأدوات في هذا المجال منها ما هو مستقل ومنها ما ينتمي لمنصة عمل متكاملة تغطي جميع مراحل الإجرائية البرمجية.

تهتم **إدارة الإصدارت** بمجموعة من المهام مثل تحديد الإصدارات وتسجيل السحوب المتوفرة من النظام:

1- وضع آلية ترقيم لإصدارات النظام.
2- التخطيط لمواعيد إنتاج إصدارات جديدة من النظام.
3- ضمان تطبيق إجراءات وأدوات إدارة الإصدارات والسحوب.
4- التخطيط للسحوب الجديدة وتوزيعها.

الإصدار هو نسخة جديدة من النظام تختلف وظيفياً عن النسخ المتوفرة سابقاً. أما عند المحافظة على الوظائف وتغيير النواحي غير الوظيفية فيسمى ذلك متغيراً **variant**. أما السحب فهو نسخة من النظام توزع على المستخدمين من خارج فريق التطوير.

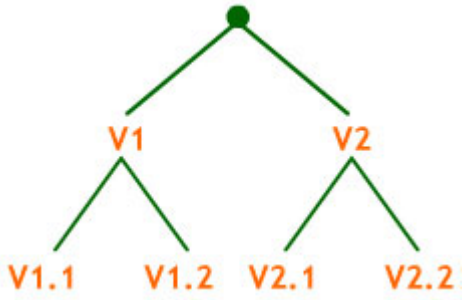
عندما يتألف النظام من مجموعة كبيرة من المكونات التي تتألف بدورها من مكونات جزئية وفرعية على أكثر من مستوى تكون هنالك حاجة كبيرة لإدارة إصدارات النظام ومعرفة مرجعية كل مكونة جزئية إلى الإصدار التي تنتمي إليه. يجب أن تعرف إجراءات تحديد الإصدارات طريقة واضحة لتحديد إصدارات المكونات. هنالك ثلاثة تقنيات أساسية لتحديد المكونات:

1- ترقيم الإصدارات.
2- التحديد بالاعتماد على الواصفات.
3- التحديد الموجه بالتغييرات.

تقنيات تحديد المكونات

1- ترقيم الإصدارات

آلية تسمية تستخدم اشتقاق خطي للأسماء ... V1, V1.1, V1.2, V2.1, ينتج عنها هرمية من الإصدارات على شكل شجرة.



2- التحديد بالاعتماد على الواصفات

يمكن إرفاق واصفات مع كل إصدار بحيث يتحدد الإصدار بمجموعة من الواصفات من الأمثلة عن هذه الواصفات: التاريخ والمبرمج ولغة البرمجة والزبون والحالة ...

يعطي ذلك مرونة أكبر من الترقيم الخطي لكنه يمكن أن يولد مشاكل في فردية النسخة. يجب أن يجري اختيار مجموعة محددة من الواصفات بحيث يمكن تحديد كل إصدار بصورة وحيدة. من الفوائد الهامة لهذه الطريقة أنها تدعم إجراء استعلامات فيمكن مثلاً البحث عن أحدث إصدار بلغة جافا من نظام ما أو إجراء استعلام من الشكل:

```
AC3D (language =Java, platform = XP, date = Jan 2003)
```

3- التحديد الموجه بالتغييرات

يجمع بين الإصدارات والتغييرات التي أُجريت للحصول عليها. يستخدم للأنظمة ولا يستخدم للمكونات. كل تغيير مقترح يتضمن مجموعة تغييرات يجب أن تُطبق بالتسلسل لتنجيزه. فتنتج إصدارات متعددة كل منها يحتوي مجموعة من التغييرات.

يجب أن تتضمن **السحوبات** جميع التغييرات خاصةً تلك المتعلقة بتصحيح أخطاء اكتشفها المستخدمون أو المتعلقة بتغيير العتاد المادي وبيئة التشغيل. كما يجب أن تتضمن وظائف جديدة للنظام. تهتم إدارة السحوب بتقرير الوقت المناسب لإصدار سحب ما (لتحويل إصدار ما إلى سحب).

تتضمن السحوبات إضافةً إلى البرامج التنفيذية ملفات التشكيلات التي تتضمن كيفية تشكيل البرمجيات وإرسائها، وملفات المعطيات اللازمة لعمليات النظام، وبرنامج إرساء، ووثائق الكترونية ومطبوعة للمساعدة، ولوازم الحزم البرمجية والدعايات المرافقة لمنتجات أخرى ذات علاقة بالتطبيق. يجري إصدار السحوبات على أقراص ضوئية (CD, DVD) أو كملفات قابلة للتحميل عن طريق الوب.

يجب أن لا تفترض إدارة السحوب أن السحوب السابقة قد جرى قبولها وتعتمد على مكونات منها لسحب جديد، بل يجب أن يتضمن السحب الجديد كافة الملفات اللازمة للإرساء. يمكن أن لا يرغب المستخدم باقتناء سحب جديد من نظام ما لكونه مكتفي بالنظام الحالي ولا يحتاج للوظائف الجديدة المتوفرة في السحب الجديد.

إن تحضير وتوزيع سحب جديد من نظام ما هو إجرائية مكلفة لذلك يجب دراسة هذه العملية جيداً واتخاذ قرار مناسب بشأنها. يمكن لعدة عوامل أن تؤثر على هذا القرار مثل: الجودة التقنية للنظام، والمنافسة، ومتطلبات التسويق، وطلبات التغيير من قبل الزبائن.

تتوفر أدوات خاصة بإدارة الإصدارات والسحوبات تسمح بترقيم آلي للإصدارات (النظام والمكونات)، وتسمح بتخزين الاختلافات بين الإصدارات، وبتسجيل تاريخي للإصدارات وأسبابها، وتساعد في فصل أعمال التطوير على الإصدارات المختلفة.

1. هندسة البرمجيات بمعونة الحاسوب CASE

يستخدم هذا التعبير للحديث عن البرمجيات التي تدعم نشاطات هندسة البرمجيات في مختلف مراحلها: من هندسة المتطلبات إلى التصميم والتطوير والاختبارات. تتضمن أدوات هندسة البرمجيات بمعونة الحاسوب محررات تصميم وقواميس معطيات ومترجمات ومفليات debuggers وأدوات بناء البرامج.

توفر تقنيات هندسة البرمجيات بمعونة الحاسوب CASE دعماً لأتمتة بعض نشاطات هندسة البرمجيات، وتهتم بأتمتة الأعمال الروتينية التكرارية ضمن نشاطات هندسة البرمجيات، فهي تدعم على سبيل المثال النشاطات التالية:

1- تطوير النماذج البيانية في مراحل تحديد المتطلبات وتصميم البرمجيات من خلال تقديم محررات بيانية مناسبة.
2 فهم التصميم باستخدام قاموس معطيات يتضمن معلومات عن الكيانات والواصفات والعلاقات في التصميم.
3 توليد واجهات الاستخدام اعتباراً من توصيفات بيانية يمكن بناؤها تفاعلياً من قبل المستخدم.
4 تغلية البرامج لاكتشاف الأخطاء باستخدام المعلومات المتوفرة عن طريقة التنفيذ.
5 ترجمة آلية للبرامج من إصدار قديم للغة برمجة ما إلى إصدار أحدث.

أدى استخدام تقنيات هندسة البرمجيات بمعونة الحاسوب CASE إلى تحسينات كبيرة في جودة البرمجيات وإنتاجيتها، لكن هذه التحسينات كانت أقل مما جرى توقعه عند بداية إنشاء هذه التقنيات، فقد كان من المتوقع الوصول إلى توفير كبير جداً في تكلفة إنتاج البرمجيات. إن التحسينات التي يمكن أن نجنيها من استخدام هندسة البرمجيات بمعونة الحاسوب CASE محدودة بعاملين اثنين:

1 هندسة البرمجيات هي في الأساس نشاط تصميم يعتمد على عمل ذهني إبداعي لا يمكن أتمته.
2 هندسة البرمجيات هي نشاط جماعي تفاعلي لا تدعمه أدوات هندسة البرمجيات بمعونة الحاسوب CASE دعماً كاملاً.

أصبحت تقنيات هندسة البرمجيات بمعونة الحاسوب CASE اليوم في مرحلة ناضجة جداً، وهنالك أدوات مقدمة من طيف واسع من الباعة والموردين.

2. تصنيف أدوات هندسة البرمجيات بمعونة الحاسوب

يساعد تصنيف أدوات هندسة البرمجيات بمعونة الحاسوب على فهم أنواعها وأدوارها في دعم نشاطات هندسة البرمجيات.

يمكن تصنيف أدوات هندسة البرمجيات بمعونة الحاسوب بعدة طرق ومن عدة وجهات نظر أهمها:

1- **المنظور الوظيفي**: حيث تُصنف أدوات هندسة البرمجيات بمعونة الحاسوب حسب وظائفها.
2- **المنظور الإجرائي**: حيث تُصنف أدوات هندسة البرمجيات بمعونة الحاسوب حسب النشاطات الإجرائية التي تدعمها.
3- **المنظور التكاملي**: حيث تُصنف أدوات هندسة البرمجيات بمعونة الحاسوب حسب كيفية تنظيمها ضمن وحدات متكاملة تدعم نشاطاً أو أكثر من نشاطات هندسة البرمجيات.

1- التصنيف الوظيفي

يظهر الجدول التالي الأنواع المختلفة لأدوات هندسة البرمجيات بمعونة الحاسوب من الناحية الوظيفية ويعطي أمثلة عن كل منها:

لا يشكل هذا الجدول قائمة كاملة بجميع أدوات هندسة البرمجيات بمعونة الحاسوب فهنالك مثلاً أدوات متخصصة بإعادة الاستخدام لم يرد ذكرها في الجدول.

نوع الأداة	أمثلة
أدوات تخطيط	مخططات PERT، أدوات تقدير، جداول الكترونية
أدوات تحرير	محررات نصوص، محررات مخططات، معالجات كلمات
أدوات إدارة التغيير	أدوات تتبع تغيير المتطلبات، أنظمة إدارة التغيير والتحكم به
أدوات إدارة التشكيلات	أنظمة إدارة الإصدارات، أدوات بناء الأنظمة
أدوات النمذجة الأولية	لغات من مستوى عالي، مولدات واجهات استخدام
أدوات دعم الطرائق	محررات تصميم، قواميس معطيات، مولدات رماز
أدوات معالجة اللغات	مترجمات، مفسرات
أدوات تحليل البرامج	مولدات الربط المرجعي، محللات ساكنة وديناميكية
أدوات الاختبار	مولدات معطيات الاختبار، أدوات مقارنة الملفات
أدوات تفلية	أنظمة تفلية تفاعلية
أدوات توثيق	برامج لأشكال الصفحات، محررات صور
أدوات إعادة هندسة	أنظمة ربط مرجعي، أنظمة إعادة بناء البرامج

2- التصنيف الإجرائي

يظهر الجدول التالي الأنواع المختلفة لأدوات هندسة البرمجيات بمعونة الحاسوب من الناحية الإجرائية من خلال المراحل الإجرائية التي تدعمها كل منها. حيث يمكن استخدام أدوات للتخطيط والتقدير ومحررات نصوص ومعدات وثائق وأدوات إدارة التشكيلات:

	تحديد المواصفات	التصميم	التنجيز	التحقق والإقرار
أدوات تخطيط	√	√	√	√
أدوات تحرير	√	√	√	√
أدوات إدارة التغيير	√	√	√	√
أدوات إدارة التشكيلات		√	√	
أدوات النمذجة الأولية	√			√
أدوات دعم الطرائق	√	√		
أدوات معالجة اللغات		√	√	
أدوات تحليل البرامج			√	√
أدوات الاختبار			√	√
أدوات تفلية			√	√
أدوات توثيق	√	√	√	√
أدوات إعادة هندسة			√	

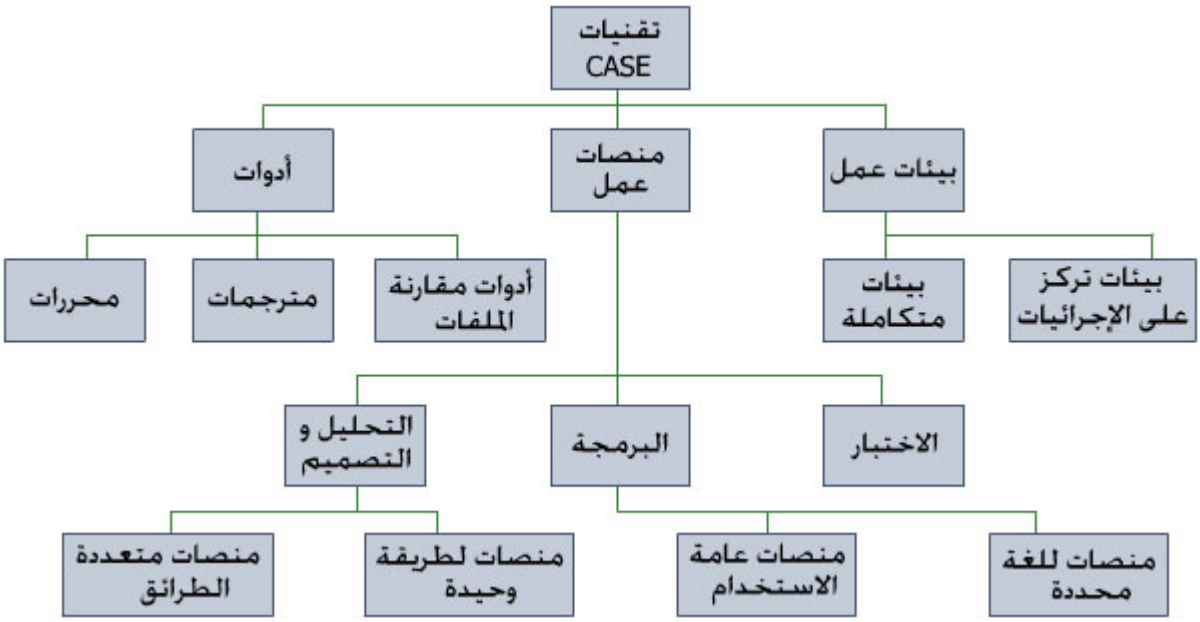
3- التصنيف التكاملي

ويعبر عن طيف الدعم الذي تقدمه تقنيات هندسة البرمجيات بمعونة الحاسوب، حيث يمكن تصنيف الأنظمة ضمن ثلاثة أنواع:

1- أدوات تدعم إجرائيات مستقلة كتفحص تناسق التصميم، وترجمة برنامج ومقارنة نتائج اختبار. يمكن أن تكون هذه الأدوات ذات هدف عام كمحررات النصوص أو تكون مجموعة ضمن منصات عمل.

2- منصات عمل تدعم مراحل دورة حياة البرمجيات ونشاطاتها من تحديد المواصفات إلى التصميم إلى التنجيز والاختبارات. وتكون عادةً مجموعة من الأدوات المتكاملة.

3- بيئات عمل تدعم إجرائية البرمجة كلياً أو جزئياً وتتألف من عدة منصات عمل.



3. أهم المنتجات المتوفرة في هندسة البرمجيات بمعونة الحاسوب

تبرز أهمية استخدام الأدوات المساعدة في هندسة البرمجيات CASE عند بناء نماذج لأنظمة معقدة. يمتد مجال المنتجات المساعدة في هندسة البرمجيات من الأدوات البسيطة جداً التي تكتفي برسم الأشكال دون فهمها (مثل Visio)، إلى الأدوات المتوسطة التي تفهم المخطط وتقوم بتفحصات بسيطة للتأكد من صحته (مثل BPWin)، إلى الأدوات الشاملة التي تتضمن إمكانات نمذجة متعددة (مثل Visible Analyst Workbench).

فعند بناء نماذج المعطيات مثلاً تسمح هذه الأدوات بإنشاء ومتابعة النماذج المنطقية والفيزيائية ثم توليد عدة أنواع من قواعد المعطيات آلياً باستخدام النماذج المعرفة، وتوليد نصوص SQL التي تتضمن تعليمات بناء الجداول والأغراض الأخرى في قاعدة المعطيات. من هذه الأدوات ما هو مخصص لنمذجة المعطيات مثل ERwin ومنها ما يجمع عدة إمكانيات نمذجة فيكامل نماذج المعطيات مع نماذج الإجرائيات والأجزاء الأخرى في المشروع مثل Visible Analyst Workbench، ومنها ما يكون مقدم من قبل بائع معين ويرتبط بنظام إدارة قواعد المعطيات الذي يختص به مثل Oracle Designer.

أما عند بناء مخططات البنية فتسمح بعض هذه الأدوات بالتأكد من جودة هذه المخططات. فالأداة Visible Analyst Workbench مثلاً تتأكد من تسمية كل المجتزآت وصحة الاتصال فيما بينها ومن درجة تعقيد الواجهات وتكامل التصميم وتعطي تحذيرات حول حالات قابلية إعادة الاستخدام والتفريع.