

الأكاديمية العربية الدولية



الأكاديمية العربية الدولية
Arab International Academy

الأكاديمية العربية الدولية المقررات الجامعية

كتاب
العربية



مدينة الملك عبدالعزيز
للعلم والتقنية KACST

هندسة البرمجيات



تأليف: جاك برينتز

ترجمة: زينا مغربل

مراجعة: د. محمد مراياتي

٢٠١٥ - ١٤٣٦ هـ - م

ماذا
أعرف؟

Que
sais-je?

كتاب
العربية



مدينة الملك عبدالعزيز
للعلوم والتقنية KACST

هندسة البرمجيات

تأليف: جاك برينتز

ترجمة: زينا مغربل

مراجعة: د. محمد مراياتي

١٤٣٦هـ - ٢٠١٥م

ماذا
أعرف؟

Que
sais-je?



www.j4know.com

٣ مدينة الملك عبدالعزيز للعلوم والتقنية، ١٤٣٦هـ

فهرسة مكتبة الملك فهد الوطنية أثناء النشر

برينتز، جاك

هندسة البرمجيات. / جاك برينتز: زينا مغربل؛ محمد

مراياتي. - الرياض، ١٤٣٦هـ

ص.: ٥٠٠

ردمك: ٩٧٨-٦٠٣-٨٠٤٩-٧٩-٢

١- البرمجيات ٢- الهندسة - برامج الحواسيب

أ. زينا، مغربل (مترجم) ب. مراياتي، محمد (مراجع) ج. العنوان

ديوي ١٠٥، ١ ١٤٣٦/٧٢٤٧

رقم الإيداع: ١٤٣٦/٧٢٤٧

ردمك: ٩٧٨-٦٠٣-٨٠٤٩-٥٩-٢

جميع الحقوق محفوظة



مدينة الملك عبدالعزيز
للعلوم والتقنية KACST

مدينة الملك عبدالعزيز للعلوم والتقنية

ص.ب. ٦٠٨٦ الرياض ١١٤٤٢

المملكة العربية السعودية

هاتف: ٤٨٨٣٤٤٤ - ٤٨٨٣٥٥٥ فاكس: ٤٨٨٣٧٥٦

الموقع الإلكتروني: www.kacst.edu.sa

إصدارات المدينة: publications.kacst.edu.sa

البريد الإلكتروني: awareness@kacst.edu.sa

رقم الإيداع الدولي للأصل بالفرنسية:

ISBN 2 13 053309 4

الطبعة الخامسة

تم الإصدار ضمن التعاون المشترك بين مدينة الملك عبدالعزيز للعلوم والتقنية

والمجلة العربية (الثقافة العلمية للجميع)



مقدمة

هندسة البرمجيات علم هندسي غايته صنع الأنظمة المعلوماتية، وهي أنظمة معقدة في أغلب الأحيان، إذ تتضمن كل ما يتعلق بمعالجة المعلومات اللازمة لحسن سير صناعاتنا، وأشكال الإدارة، والاتصالات، وأنظمتنا الدفاعية - أي بإيجاز - منظومتنا الاجتماعية والاقتصادية بأسرها.

ويتكون النظام المعلوماتي من مجموعة من الحواسيب، المتنوعة المصدر والقدرة، والمترابطة فيما بينها بواسطة شبكات محلية (الشبكات المؤسسية الداخلية) وشبكات خارجية (الشبكات المؤسسية البينية)، فضلاً عن أجهزة طرفية متعددة (جهاز صرف آلي، رادار، إنسان آلي...)، من شأنها استقبال المعلومات وإعادة إرسالها ضمن بيئتها.

ويتألف نظام من هذا القبيل مما يلي:

- **العنصر المادي - المعدات** (مثل الحواسيب والطرفيات وأجهزة المودم المختلفة، وأجهزة توزيع الشبكة، وأجهزة المكشاف والمشفل إلخ...) التي تتولى توليد الطاقة الإجمالية لمعالجة وربط النظام بالعالم الخارجي.
 - **العنصر المنطقي - البرمجيات** المسؤولة عن الوظائف المنطقية اللازمة لأداء شتى عمليات معالجة وحفظ المعلومات.
- وتأتي هذه البرمجيات في ثلاثة أنواع:
- **البرمجيات الخاصة بُمَصْنَعِ المعدات** وهي بالغة الارتباط بالمعدات؛
 - **حزمة البرمجيات** التي يصيغها مطورو البرمجيات، والتي تضمن القيام بوظائف محددة، وهي أشبه بصناديق سوداء أي لا يسمح للمستخدم بالدخول إليها وتعديلها ولكن تستخدم عن طريق متغيرات وسيطة.
 - **البرمجيات المطورة خصيصاً لخدمة أغراض مؤسسات الاعمال،**

سواء من قبل المؤسسة ذاتها أو عن طريق شركات الخدمات.

وتعنى هندسة البرمجيات بعمليات تصنيع البرمجيات المختلفة بشكل يضمن:

- تلبية المنتج المُصنَّع لاحتياجات صاحبه بشكل صحيح ودقيق (وهو المالك أي العميل النهائي)؛
- اقتصار التكاليف ومدد التنفيذ، على الحدود المرسومة منذ البداية؛
- الالتزام بعقد الخدمة (من حيث الأداء وسلامة التشغيل، والأمن، إلخ...) طيلة استخدام برنامج الحاسوب لاحقاً.

وكما يبين مصطلح software (سوفتوير)* باللغة الانجليزية، يمكن اعتبار البرمجيات الجزء اللين أو المرن من النظام الذي نستطيع تعديله كما نشاء، وفق مواطن الحاجة المستجدة طيلة استخدامنا للنظام. وتعد هذه القدرة على التطور سمة مميزة باللغة الأهمية للبرمجيات، لأنها بمثابة العنصر المحدد لعمر النظام (المرونة وقابلية التكيف وسهولة الاستخدام)، وبالتالي فهي العنصر المحدد لتكلفة الاستهلاك.

وتهتم هندسة البرمجيات بما «وراء البرمجيات»، فهي ترسي عدداً من القواعد الثابتة والضمانات، التي تكون بمثابة شروط تضمن سير عملية التصنيع على نحو سليم. وكسائر العلوم متعددة التخصصات أو «تخصصات الميتا Metadiscipline» يتأصل علم البرمجيات في عملية تطوير البرمجيات، مع البحث المستمر عن شروط فعاليته ومدى ملائمة الأدوات التي يطرحها. وتقادياً لعقمه مع الزمن، لا بد لهذا العلم من القيام باستمرار بتحدي الأنماط المتقادمة، والمصطلحات التقنية التي كثيراً ما تخفي فراغاً فكرياً.

ولا يمكن عرض علم زاخر بهذا الشكل في حوالى ١٢٨ صفحة أنظر المراجع [٧،٨،٩] إلا من خلال عملية تبسيط جذرية لمسائله، والاكتفاء بالوقوف عند بعض السمات الأساسية المتفق عليها والتي سنحرص قدر المستطاع على تقديمها بأسلوب حدسي.

* يستعمل مصطلح «الكيان اللين» لدى بعض دول المشرق العربي.

الفصل الأول

بعض المعطيات الاقتصادية

١. تكلفة البرمجيات

تقاس تكلفة البرمجيات عادةً بوحدة أفراد- شهور (فش) أو أفراد- أعوام (فع) التي ينبغي تمييزها عن مدة التطوير، فعمل ٣ مهندسين مدة ١٨ شهراً تساوي تكلفة قدرها ٥٤ فش أو ٥,٥ فع.

أما حجم البرنامج الحاسوبي، فيقاس عادةً بعدد أسطر شفرة البرنامج المصدر أو التعليمات (سطور المصدر (سم)) أو ألف* سطر من المصدر (كسم)، التي يتضمنها برنامج الحاسوب الذي جرى تسليمه والجهاز للاستخدام. وعامل الحجم هذا، الذي يقيس الجزء القابل للتنفيذ في أي جهاز، هو الذي اعتمد كمؤشر رئيسي لكم المعلومات التي تحملها البرمجية. من هنا تقاس إنتاجية التطوير البرمجي بـ (سم) لكل (فش). ولعل هذا المؤشر هو أبغ ما يدل على مدى صعوبة صنع البرمجيات أو سهولتها. ويبين الجدول التالي بعض الإحصائيات ذات الدلالة. وقد تبدو هذه البيانات مبهمة إلى حد ما، إلا أنه بمقارنتها ببعض أمور الحياة اليومية، يمكن أن نلمس ما تنطوي عليه من تعقيد. فبرنامج حاسوبي بقياس ١٠٠ كسم على سبيل المثال يعادل بمرفقاته حوالي ١٠ كتب يتألف كل منها من ٤٠٠ صفحة.

نوع البرمجيات	التكلفة	الحجم	ملاحظات
مصرفات لغات برمجة: compilers			النطاق الزمني (المدة اللازمة لتطوير المنتج):
-باسكال، سي	١٠ فع	٢٠-٣٠ كسم	عام إلى عامين
-كوبول، فورتران	٨٠-١٠٠ فع	١٠٠-٢٠٠ كسم	٢-٣ أعوام
-أدا (Ada)	١٥٠-٢٠٠ فع	< ٣٠٠ كسم	< ٣ أعوام

* ألف = كيلو.

نطاق زمني من ٣ إلى ٥ أعوام بما في ذلك نسخة تجريبية مبدئية	٦٠٠-٣٠٠ كسم	٥٠٠-٣٠٠ فع	أنظمة إدارة قواعد البيانات العلائقية (أوراكل، دي بي ٢٠٠)
			أنظمة لحظية (زمن حقيقي) كبيرة:
نطاق زمني ٦ سنوات، معد بلغة هال	٢٢٠٠ كسم	< ١٠٠٠ فع	- مركبة فضائية
نطاق زمني ٧ سنوات معد بلغة من نوع بي ١١	٢٢٦٠ كسم	٥٠٠٠ فع	- سيفغارد ^١ SAFEGUARD
نطاق زمني ١٠ أعوام ومدة وسطية قبل حدوث عطل قدرها ٥٥ ساعة ^٣	٩٦٠ كسم	٩٥٥ فع	- ساير ^٢ SABRE
مدة حياة البرنامج من ١٥ إلى ٢٠ سنة، بما في ذلك ما لا يقل عن ٥ سنوات أمد النسخة المبدئية	٥٠٠٠-١٠٠٠٠ كسم (معد بلغة برمجية متقدمة غالباً منذ السبعينات)	٥٠٠٠-٢٥٠٠ فع	أنظمة تشغيل مصنعة إم في إم إس MVS، في إم إس VMS، جي سي أو إس 7 GCOS7
باستخدام لغة كوبول / إل ٤ جي	١٠٠٠-٥٠٠ كسم- قواعد بيانات بالغة الأهمية	٥٠٠-٣٠٠ فع	أنظمة صناعية: جي بي إيه أو GPAO، إم آر بي MRP، سي أي إم CIM، ...
تتطوي على خوارزميات رقمية متقدمة	بلغة فورتران في الغالب < ٢٠٠ كسم	٣٠٠-١٥٠ فع	البيانية (غرافيك) (بعدان، ٣ ابعاد)

يتطلب استخدام هذه اللغات البرمجية في بيئة صناعية تطويراً أمثلاً شاملاً	بلغة سي في الغالب مع طرائق تفسير متقدمة		الذكاء الاصطناعي
		١٠ - ٢٠ فع	ليسب LISP ، برولوج PROLOG
		٢٠ - ٣٠ فع	- الأنظمة الخيرة
تتضمن طيفاً من الأدوات التي تشمل مراحل دورة التطوير	مفصلة استناداً إلى قاموس	١٠٠ إلى ٢٠٠ فع	أدوات هندسة البرمجيات بمعونة الحاسوب

وَيُبْرَزُ عِظَمُ هذه الأرقام، وهي لا تتضمن تكلفة الصيانة المتكررة، دَوْرَ هندسة البرمجيات وضرورتها، فمن شأن أي تحسين لعملية صناعة البرمجيات أن يسفر عن مكاسب ملموسة في الإنتاجية. إلا أنه عند إضافة عامل عمر المنتج في البرمجيات، تظهر عقبتان جديدتان:

- التغييرات التي لا مفر من إجرائها على هذه البرمجيات والمتعلقة بعمرها.

- التحديات التنظيمية الخاصة بتنقل طاقم العاملين بين الشركات وحتى ضمن نفس الشركة،

هذا وتزداد قيمة التكاليف ذات الصلة كلما طالت الحياة المتوقعة للبرنامج (التكاليف المتكررة). كما تتأثر هذه التكاليف بمستوى بنية البرنامج الحاسوبي المبدئي، لذا فمن الأهمية بمكان النظر إلى هذه التكاليف من منظور شامل:

تطوير + صيانة + تشغيل وحتى العزوف عن استخدامه.

٢. هيكل التكاليف

تجدر الإشارة إلى التغيير الكبير الذي طرأ على هيكل التكاليف الحاسوبية، إذ كانت تكاليف العناصر المادية أي المعدات تمثل القسط

١ - نظام دفاعي مضاد للصواريخ الباليستية في الولايات المتحدة في حقبة السبعينات
٢ - نظام حجز خاص لشركة أميركان إيرلاينز من إنتاج شركة أي بي إم
٣ - وحدة قياس الأداء الجيد

- الأعظم في السابق، إلا أن الأمر تغير على مرحلتين:
- سمحت سياسة «التفكيك» التي فرضتها شركة أي بي إم بتحليل تكلفة البرمجيات على هذا النحو أي جزء جزء.
 - سلط ظهور الحوسبة الدقيقة أو الصغرى* في حقبة الثمانينات الضوء على انخفاض التكلفة للعناصر المادية بشكل غير مسبوق منذ ولادة المجتمعات الصناعية.
 - وقد أتاح تدني أسعار الأجهزة الحاسوبية على هذا النحو افتتاح المعلوماتية لجميع أصعدة الحياة الاقتصادية. ونتيجة لهذا التطور السريع - حوالي ١٠ أعوام - بات أسلوب صنع البرمجيات في طليعة الأولويات.

٣. تطور الطلب

- تنامى الطلب على جميع أنواع البرمجيات بشكل ملحوظ في حقبة الثمانينات، وهي ظاهرة ارتبطت بعدة عوامل إيجابية هي:
- طلب هائل على التطبيقات الجديدة.
 - تفاعل متنام ما بين المستخدم والأنظمة المعلوماتية.
 - رقمنة كل ما كان تناظرياً analogae في السابق مثل الكتب والموسيقى والصور والأفلام والإشارات الصوتية واللاسلكية.
 - زيادة الخدمات المعلوماتية أو الحاسوبية المطلوبة حجماً وعدداً، وبخاصة في مجال إدارة الأنظمة المعقدة.
 - ظهور مجالات جديدة ذات أهمية، مثل ألعاب الفيديو والرسومات الحاسوبية.
 - الأمر يرقى إذن لكونه ظاهرة شاملة تسفر عن طلب هائل على التقنيات والوسائل التي تسمح بالإنتاج بسرعة متزايدة، مع تعزيز مستوى الجودة.

* micro informatique

وتزامناً مع هذه الظاهرة، أصبحت البرمجيات عنصراً أساسياً في الأنظمة ذات الوظائف الحرجة، حتى باتت سلامة ووثوقه أداء البرمجيات تشكل التحدي الأكبر الذي يواجه هندسة البرمجيات. فلنقف عند بعض نماذج المخاطر الممكنة:

- المخاطر البشرية: أوامر قيادة الطائرات المدنية، والسيطرة على حركة المرور الجوية، والقطارات بالغة السرعة، إلخ.
- المخاطر الاقتصادية: حوسبة مداولات أسواق الأسهم، المقاسم المركزية الهاتفية المحوسبة، الصرافة الالكترونية، إلخ.
- المخاطر الاجتماعية: الأنظمة التي لا تلبى احتياجات المستخدمين والتي تولد ردود فعل سلبية، وسرية المعلومات التي يحفظها النظام، وأوجه الجروح الجديدة من قبل المستخدمين غير الأمناء (برامج الفيروس وشفرات حصان طروادة والقنابل الذكية) الساعين لاستغلال النظام لخدمة مآربهم.

وتجدر الإشارة هنا إلى ما تمثله سلامة البرمجيات من تحد هائل. فخلافاً للمجالات الهندسية الأخرى، لا تتم الأخطاء أو مواطن الضعف هنا عن عيب في المواد المستعملة أو ظاهرة استهلاكية معروفة القواعد، وإنما تتبع من أخطاء بشرية متأصلة في عملية البرمجة ذاتها.

لذا فإن من أبرز غايات هندسة البرمجيات تصميم عمليات تصنيع تكفل سلامة هذه البرمجيات من جميع هذه لأخطاء - أو على الأقل - إن لم يكن هذا ممكناً كما هو الحال في معظم الأحيان، التأكد من إمكانية معالجة تداعيات الأخطاء المتبقية - والتي نجهل عددها - من خلال عمليات مخصصة لهذا الغرض، ضمن إطار زمني يتفق ومهمة النظام. وفيما يتعلق بالأنظمة اللحظية (الزمن الحقيقي)، فقد يكون هذا الإطار الزمني قصيراً جداً (بضعة أجزاء من الألف من الثانية).

ومما يزيد هذه الغاية تعقيداً، افتران ظاهرتين حديثتين:

- زيادة قدرة المعدات أو الأجهزة الحاسوبية بشكل هائل يسفر سلباً

عن زيادة تواتر هذه الأخطاء.

- التعميم التدريجي للحوسبة الموزعة على شبكات حاسوبية، تسمح بجمع عدة تطبيقات بمنتهى السهولة، بمساعدة نماذج برمجية، على غرار التعاملات الموزعة أو مخدم-زبون. من شأن مثل هذا "الاقتران" الذي يعزز حالات إصابة أو خلل الأنظمة، تعقيد مهمة تشخيص نوع العطل، بل وجعلها مستحيلة أحياناً في غياب المرشحات البرمجية.

نتيجة لجميع هذه التطورات، بات القسط الأعظم من تكاليف البرمجيات يُنفق في عمليات التأكد والتحقق من سلامتها.

* نظام شبكة حاسوبية يقوم على مبدأ وجود حاسب مخدم Serveur، وحواسيب زبائن موصلة عليه Clients.

الفصل الثاني

تصنيف البرمجيات

١. أوجه الصعوبات

كنا، ولزمن طويل، نميز بين ما يسمى بالمعلوماتية الإدارية، والتي يفترض كونها بسيطة، والمعلوماتية العلمية أو التقانية، التي يفترض كونها أكثر تعقيداً. إلا أن هذه المقارنة لم تعد تصلح اليوم في ظل انتشار الشبكات والأنظمة الموزعة. وقد حاولنا في السابق أيضاً تصنيف البرمجيات وفق مجالات التطبيق الكبيرة التي يفترض أن تعكس درجات متفاوتة من التعقيد.

المجال	هيكل البيانات	الخوارزميات	التحكم
الإدارة	معقد	بسيط	بسيط*
التحليل الرقمي، المحاكاة	بسيط	معقد	بسيط
الاتصالات	بسيط	بسيط	بالغ التعقيد
الزمن الحقيقي (الأنظمة اللحظية)	بسيط	معقد	معقد
معالجة وتحليل البيانات	بالغ التعقيد	معقد	بسيط
أنظمة التشغيل، المصرفات	معقد	معقد	معقد

* باستثناء نموذج المخدم- الزبون

وبعيداً عن قيمة هذا التصنيف الوصفية، فهو لا يخدم المصمم لأنه لا يسهل عمله ولا عملية اتخاذ القرار. إذ ينبغي أن يعكس التصنيف الفعال ما يكشف عن تحدٍ حقيقي نستطيع الإعداد له فيما يتعلق بـ :

الأفراد المناسبين والتنظيم الجيد؛

الأساليب المناسبة؛

الأدوات المناسبة؛

لقد اتضح أهمية وجود القيود التي تفرضها البيئة أو عدم وجودها (المراجع ١٧، ٢٠، ١) في عملية تصنيف البرمجيات. إذ يمكن تحديد ثلاثة مكونات رئيسية مميزة لثلاثة أنواع من البرامج نجدها في جميع المجالات التي أشرنا إليها آنفاً:

- البرامج الحاسوبية ذات المواصفات المحددة والثابتة، والتي نشير إليها بالبرامج التي من نوع ث، أو ث— برمج*.
- البرامج الحاسوبية التي تعالج عدداً يحتمل أن يكون لا متناهياً من المسائل المتماثلة، والتي تتضمن معايير يمكن أن تتغير وفق احتياجات المستخدم، فهي برامج من نوع م أو م— برمج؛
- البرامج التي تتفاعل مع البيئة (الأفراد، أجهزة الاستقبال، الأنظمة الأخرى، إلخ...) والتي ربما تكون تفاعلات عشوائية، وذات مدد زمنية مختلفة، ويحتمل أن تشوبها الأخطاء، وهي البرامج من نوع ب أو ب— برمج.

فلنعرض مثلاً على كل نوع من هذه البرامج:

١. **ث- برمج-** فلنفترض أننا نريد تصميم برنامج يقوم بحساب

قيمة الجذر التربيعي للعدد ما A . هناك العديد من الطرق

المدونة للقيام بهذه العملية الحسابية، مثل طريقة نيوتن:

$$\sqrt{A} = \lim_{n \rightarrow \infty} x_{n+1} = \frac{1}{2} \left(x_n + \frac{A}{x_n} \right) \text{ avec } x_0 = \frac{A}{2}$$

* الأنواع الثلاثة هي: E-prog، S-prog، P-prog

تعد هذه المعادلة على أنها مواصفة البرنامج، وهي صحيحة في جميع الحالات المحتملة ولا تتعلق ببيئة استخدام البرنامج. وتقتصر البرمجة المنفذة لهذه المعادلة على مجرد ترجمة بسيطة لها تراعي حدود الجهاز (دقة الأرقام، معيار تقارب هذه السلسلة، إلخ...). وثمة الكثير من البرامج التي تندرج ضمن هذا النوع، كما تتوافر خوارزميات كثيرة ومتباينة لحل طيف واسع من مثل هذه المسائل (المرجع ٣). ويكفي للتحقق من صحة هذا البرنامج التأكد من قيامنا بترجمة المعادلة على نحو سليم (فقط لا غير)، وربما إثبات ذلك، دون أية آثار جانبية.

٢. م-برمج- فلنفترض الآن أننا بصدد تصميم برنامج يقوم

بترجمة نص مكتوب بلغة البرمجة أدا (Ada) إلى نص مقابل في لغة الآلة (يسمى مثل هذا الجهاز بالمُصَرِّف أو المترجم البرمجي). ينبغي أن يتمكن المصرف من معالجة عدد لا متناه من النصوص، علماً بأن قواعد الكتابة محدودة، لأنها قواعد عودية. بل إن على المصرف ترجمة جميع النصوص الواردة إليه بنفس السرعة على الأقل. هناك عدة أنماط لاستخدام المصرف:

في مرحلة تنقيح البرنامج، يفضل المستخدم: سرعة ترجمة قصوى، وتشخيصاً وافراً ودقيقاً للأخطاء، وآليات استئناف العمل رغم الأخطاء بحيث لا يضطر الجهاز للقيام بالترجمة من جديد بعد اكتشاف خطأ وبالتالي القيام بالترجمة عدة مرات تساوي عدد الأخطاء الواردة، كما يفضل قيام المصرف بوضع خريطة البرنامج في الذاكرة، إلخ...؛ وفي المرحلة النهائية، ولإنتاج البرنامج وتجهيزه للعمل، نزيل خيارات التصحيح ونعيد إجراء عملية الترجمة بخيارات تحسين شاملة، بحيث يتصف البرنامج قدر الإمكان بالفعالية الاقتصادية بالنظر إلى موارد الجهاز الحرجة (استخدام وحدة المعالجة المركزية، حجم الذاكرة، مرات نداء مكتبات النظام، إلخ...).

على هذا النوع من البرامج إذن القيام بخيارات متناقضة، وسيكون على مصمم المُصَرِّف البحث عن حل وسط وفق خصائص النص المطروح للترجمة (طوله، مستوى مراكية البنى وتعقيدها إلخ...) ونمط الاستخدام. وهنا تكون عملية تنسيق واختيار الخوارزميات (هندسة المُصَرِّف) للقيام بالترجمة على نحو فعال، أكثر تعقيداً من تلك في النموذج السابق (ب-برمج)، وتتطلب تفعيل استراتيجيات اختيار الأمثل. كما أن عدد الحالات المحتملة للمصرف هي أضعاف ما هي عليه في النموذج السابق، وبالتالي فإن عمليات التحقق والمصادقة أكثر طولاً. أضف إلى كل ما سبق ذكره إمكانية تغيير معايير اختيار الأمثل وفق مواصفات المعدات.

وتعد هذه الحالة نموذجية بالنسبة لبرامج من نوع م—برمج: إذ نجدتها كلما حاولنا اختيار الحلول المثلى التي تقتصد في الموارد اللازمة والتي يحددها حجم وهيكل البيانات المدخلة.

٣. ب- برمج أما للبرامج التي تنتمي إلى هذه الفئة، والتي تتفاعل وتستجيب لمحفزات واردة من البيئة التي يعمل فيها البرنامج. فينبغي البدء بوصف بيئة أو محيط البرنامج، بحيث نرسم خطاً فاصلاً بين كل ما ينتمي لهذه البيئة، وما لا ينتمي إليها. وتحديد هذا الفاصل أمر بالغ الدقة والحساسية عندما تتضمن مثل هذه البيئة مشغلين آدميين أو معدات معقدة.

توصيف متطلبات هذه الأنظمة ووظائفها لا يمكن اعتباره نهائياً إذ يتطور باستمرار، ونتيجة لذلك تكون مفردات الترجمة التقنية لهذه المتطلبات والمواصفات الوظيفية ذات طابع يتطور باستمرار.

وفي سبيل الوصول إلى مرحلة التنفيذ، لا بد من اعتماد توصيف محدد، ولكن ينبغي على الأرجح إعادة النظر فيه في وقت لاحق.

وينبغي أن تُوفَّق البرمجيات المطورة لمثل هذه الأنظمة ما بين:

- استقرار النموذج وبين قدرته على التعديل والتطوير بنفس وتيرة تطور البيئة التي يعمل بها؛

- الزمن اللازم لإعادة إنتاج النسخ المتتالية وبين تواتر طلبات التغيير. إن برنامجاً حاسوبياً للتحكم بالمرور الجوي، أو نظام حجز المقاعد (الطائرات، القطارات، إلخ...)، أو برنامجاً حاسوبياً للتحكم في مقسم هاتفي مركزي، أو واجهة (أفراد-أجهزة) لمقصورة طيار، أو غرفة قيادة وتحكم... جميعها نماذج لبرامج يعد عامل البيئة فيها عاملاً سائداً.

لذا فإن التحقق من سلامة مثل هذه الأنظمة والمصادقة على حسن عملها أمر بالغ الصعوبة والتكلفة، إذ أن هذا التحقق وهذه المصادقة هي في الواقع بروتوكولات حقيقية.

وسنجد على الصعيد العملي أنظمة يكون العنصر البرمجي فيها نسيجاً مختلطاً من البرامج من صنف ث، وم وب. أما الوزن النسبي، من حيث التكاليف، مقاسةً بعدد سطور شفرة ث، وم وب فتزداد حَسَبَ النسب ١، ٢، ٤ (كما أن حجم البرنامج قد يعزز هذه الفوارق).

٢. أنماط التوزيع

ثمة سمة قوية أخرى للبرمجيات تتبع عن نمط التوزيع الخاص بها. إذ يمكن تمييز:

- برمجيات التسليم الجاهز أو الجاهزة للاستخدام، أي المصممة لتلبية غرض محدد والتي لا تتوفر إلا بأعداد محدودة جداً؛
- حزم البرمجيات التي تلبى أغراضاً عامة، والتي توجد آلاف، بل ملايين النسخ منها في تطبيقات المعلوماتية الدقيقة أو الصغيرة.

في الحالة الأولى، تنطوي تكلفة برنامج الحاسوب على تكلفة التطوير والصيانة بشكل أساسي، الناجمة عن الخصائص الفنية المحددة الخاصة ببرنامج الحاسوب، وعن إنتاجية فريق العمل الذي يقوم بتطويره. ويعد

هذا النوع من التطوير نموذجاً للمشاريع التي يكون فيها الابتكار التقني ذا أهمية (البحث عن ميزة تنافسية) والمخاطرة عالية (مواعيد التنفيذ والأداء وسلامة التشغيل).

أما في الحالة الثانية، فيكون إجمالي تكلفة برنامج الحاسوب هو مجموع عاملي تكلفة ثانويين:

- تكلفة التطوير-الصيانة، كما الحال بالنسبة لبرمجيات التسليم الجاهز:

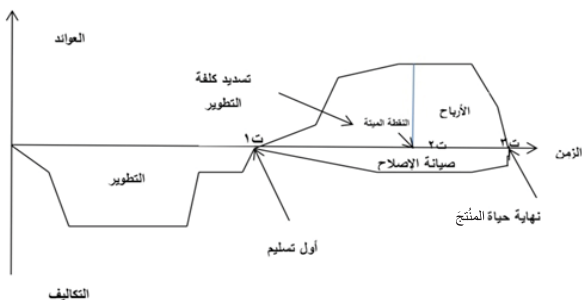
- تكلفة التوزيع التي تعتمد أساساً على تركيبة بيئة الأنظمة التي تجري متاجرة حزمة البرمجيات فيها، وعلى إجراءات الأمان التي تتضمنها حزمة البرمجيات. وهذه التكلفة هي دالة أو تابع يزداد طردياً مع تعقيد بيئة تقنية المعلومات المقامة.

ويتم تمييز فرصة المتاجرة من قبل عدد كبير من المنافسين الذين سيحاولون الاستئثار بحصص السوق، خاصة عندما تلبي حزمة البرمجيات هذه حاجة عامة، الأمر الذي يسفر عن نتيجتين:

- وجود سعر السوق- فكلما كبر حجم السوق، كلما انخفضت الأسعار وإن لم تتغير تكاليف التطوير. كما أن المستخدمين سيميلون إلى استخدام معدات أخرى في حال تم توحيد الواجهات بشكل معياري (مثل يونيكس، ويندوز إلخ...)

- ضرورة وجود عمر محدد لحزمة البرمجيات. ففي حال لم يعد المحرر نفسه لإصدار نسخة ثانية وثالثة و... إلخ من حزمة البرمجيات تتضمن وظائف جديدة لتلبية المتطلبات المستجدة بعد ظهور النسخة الأخيرة، سيكون المنافسون الآخرون قد خططوا لذلك بدلاً منه، وتغدو نسخته متأخرة عفى عنها الزمن (في الرسم التالي، ت³ لا يتوقف على المورد وإنما على المنافسين).

ويأخذ المنحني النموذجي الخاص بالتكاليف/العوائد لحزمة برمجيات الشكل التالي:



يتيح لنا هذا المنحني فهم آثار ظاهرتين متكررتين في علم المعلوماتية:

- تأخر التسليم

- الافتقار إلى الوثوقية

ففي حال تأخر التسليم، تتحرك النقطتان ت ١ وت ٢ نحو ت ٣ الثابتة.

فيقل الربح، ولا تكون ربحية تطوير المنتج عندئذ مضمونة.

وفي حال تم الالتزام بموعد التسليم، وهو ما يكون عادة على حساب مستوى الجودة، تزداد صيانة الإصلاح وتستحوذ على فريق التطوير الذي لا يعود قادراً على إعداد النسخة التالية من المنتج، فيكتسب المنتج سمعة تجارية سيئة، فتتدهوى حصة الشركة المنتجة من السوق، ما قد يكون بمثابة ضربة قاضية على مطور البرمجيات.

لا بد إذن بالنسبة لحزمة البرمجيات من الحرص على التحكم بتاريخ التسليم وكذلك وعلى الأخص جودة البرنامج، فالسوق دائماً ذات ردود فعل قاسية.

أما بالنسبة للبرمجيات من نوع التسليم الجاهز أو المفتاح باليد المطورة خصيصاً لخدمة أغراض المؤسسة المحددة، فإن التحكم بالتكاليف ضرورة قصوى، وإن أفضى ذلك إلى تأخير مواعيد التسليم. فهذا النوع من المشاريع يقتضي وجود علاقات متميزة ما بين الطرف المتعاقد والمقاول الرئيس.

الفصل الثالث

القضية الأساسية في هندسة البرمجيات

١. هدف هندسة البرمجيات

تنطوي هندسة البرمجيات على إجمالي الوسائل التقنية والصناعية والبشرية التي ينبغي جمعها لتحديد وتشديد وتوزيع وصيانة برمجيات تتصف بما يلي:

- موثوقة، بمعنى أنها تستجيب بشكل محدد لجميع الطلبات، والتي ربما تشوبها الأخطاء، المتعلقة بوظيفتها؛
- سهلة الاستخدام، أي أنها معدة لإمكانيات المستخدم الواقعية لا المفترضة؛
- قابلة للتطور، أي متكيفة مع الحاجات المستجدة ضمن فترات معقولة؛
- اقتصادية، أي أنها تحقق المعادلة المثلى ما بين الخدمة التي يتم تقديمها (الوظائف المتوافرة، والعائد على الاستثمار، وعقد الخدمة، إلخ...) وتكاليف التطوير-الصيانة المعلنة في بادئ الأمر.

٢ - طبيعة الأخطاء

- بالنظر إلى عميلة تطوير البرمجيات، نلاحظ ما يلي:
- وجود أخطاء عديدة بشكل دائم في البرمجيات، حتى بعد خضوعها لدورات مكثفة للتحقق من سلامة العمل والمصادقة عليه، قد تناهز تكلفتها ٥٠% من إجمالي التكلفة، وذلك حتى بعد شهور بل سنوات أحياناً من الاستخدام السليم؛
 - الصعوبة البالغة التي ينطوي عليها تطوير البرمجيات، ذات حجم معين، خلال فترات زمنية تتناسب مع تكاليف تطويرها، لأن التغييرات المتلاحقة حينئذ تقوض البنية المنطقية (البنيان)

للتصميم الأصلي على نحو يضعف النظام ويضطر إلى إجراء أوجه تنقيح مكلفة خُدْجية (غير ناضجة).

وينبغي تحري مصدر هذه الصعوبات في:

- الطبيعية البشرية التي يتسم بها نشاط تطوير البرامج الحاسوبية؛
- طبيعة البرمجيات العميقة.

هذا ويقتضي نشاط تطوير البرمجيات عملية تعلم وتمهن في بعض أكثر العمليات صعوبة، منها على سبيل المثال:

- العملية المنطقية الرياضية، ذلك أن كل برنامج هو محصلة عمليات تفكير واستنتاج واستقراء خاصة بكيانات نظرية بحتة غير محددة بشكل كامل. ويبدو أن القدرة الإنسانية لا حدود لها في إبداع كيانات، بيد أنه لا بد لهذه الكيانات، كي تتمكن من التحكم بها على نحو متسق باستعمال أجهزة مجردة من القدرة على التفكير، أن تكون صحيحة المنطق. لا بد إذن من إرساء قواعد مترقعة من شأنها تأطير القدرة الخيالية لدى المبرمجين؛

- بناء الأفكار التجريدية والقدرة على التعميم، ما يعد ضرورة لاستنتاج الكيانات المنطقية المنبثقة من العالم الحقيقي الذي سنقوم بتمثيله والتحكم به بواسطة البرامج؛

- توصيف المعلومات ونمذجة حركة البيانات التي هي المكونات الرئيسية للنماذج القابلة لتمثيل العالم الحقيقي، بما في ذلك جوانبه الديناميكية؛

- القدرة على الترجمة من "النظام الرسمي" إلى آخر، إذ سيكون على المبرمج إجراء تغييرات عديدة لشفرة البرنامج وفق الخوارزميات المستخدمة؛

- البراعة في التواصل الإنساني، إذ أنه من الضرورة بمكان أن نفهم ونفهم دون لبس؛ إذ سيتم تنفيذ البرنامج بمجرد وجوده في الحاسوب بشكل "غبي" ولن يكون بمقدور أي مشاهد أو مستخدم

أن يفهم المقصود من مخرجات البرنامج إن كان هناك لبس محتمل؛
 - الإحاطة بجانب التعقيد المرتبط بطبيعة البرامج وهي طبيعة كثيرة
 الاحتمالات (أو التوافقية)، وكذلك إدارة الوقت البشري (أي الوقت
 اللازم لتفاعل المستخدم مع الحاسوب) الذي يختلف عن وقت
 الجهاز (وحدة مركزية، وأقراص، وشبكات، والمعدات الطرفية التي
 لكل منها ساعتها الخاصة بها).

تشكل جميع هذه الدروس جوهر المشكلة أو التحدي الذي يمثلته تطوير
 برنامج ما، والذي يولد فعلياً الأخطاء؛ فالجميع يخطئ هنا بما في ذلك
 الأفضل.

ثمة خطأ بشري دائماً وراء كل إخفاق برمجي، كالسهر عن الأخذ
 بإحدى الحالات الخاصة، أو التحليل المغلوط لمعلومة ما، أو الاعتماد
 على نظرية لم تثبت صحتها، أو مواصفة تقتقد للترابط المنطقي، إلخ...
 ويُترجم الخطأ البشري إلى خلل في البرنامج ربما يبقى كامناً دون ظهور
 آثاره لفترة طويلة. وإذا ما اجتمعت عدة ظروف تنفيذية مع بعضها
 بعضاً وأنتجت الخلل الكامن فإن هذا الخلل ربما يسبب خطأ في تنفيذ
 الوحدة الوظيفية الخاصة بالبرنامج الذي انكشف فيه العيب (والذي ليس
 بالضرورة البرنامج المشوب بالخلل) وأخيراً، وحسب طبيعة الخلل، ربما
 يطرأ إخفاق أو عطل يسفر عن تعطل النظام تماماً. من هنا فإن تداعيات
 الخلل تكون:

- حرجة، إن عَطَلَ المهمة التي يقوم بها النظام؛
- خطيرة، إن تسبب في إجابات خاطئة وحط من عقد الخدمة؛
- محدودة، إن تسبب في إزعاج المستخدم أو هدر الوقت؛
- مقبولة، إن استطعنا التفاوضي عنها وإن كان الضيق منها ظاهراً.
- ثمة أمور هنا من الضروري الإحاطة بها:
- قد تطول المدة المنقضية ما بين إحداث خلل ما واكتشاف هذا الخلل
 (عدة أعوام)؛

- لا يسفر كل خلل بالضرورة عن إخفاق؛
- قد "يغيب" الخلل وسط حجم هائل من المعلومات السليمة بحيث يتعذر الكشف عنه؛
- حال حدوث عطل ما، ربما تكون قد مرت فترة زمنية ما من حدوث الخطأ وحتى تعطل النظام، الأمر الذي قد يعسر تمييز الوحدة البرمجية المشوبة بالخطأ نظراً (لوجود العديد من الأمور المسببة الممكنة)، والأسوأ من ذلك احتمال تسبب العطل بإتلاف لبعض المعلومات التي سيتوجب العثور عليها ومن ثم إعادتها إلى حالة من الاتساق.
- وتبعاً لهذه الظواهر المختلفة ينتج ما يلي:
- بعد تصحيح الأخطاء الأولية، سرعان ما يصبح الكشف عن الأخطاء المتبقية أمراً بالغ الصعوبة (إذ تكون العلاقة ما بين الإخفاق والخلل اقل فأقل وضوحاً)؛
- ربما تسفر معالجة الخلل عن تتالي مواطن التصحيح في وحدات أخرى تعتمد وظيفياً على الوحدة أو الوحدات التي يشوبها الخلل (يصبح أثر الخلل منتشرراً وقادراً على التعاظم)، الأمر الذي يقتضي اختباراً جديداً للبرنامج بأسره.

٣- طبيعة البرمجيات

يهتم المهندس، في مجالات الهندسة التقليدية، بتشكيل وصقل مادة طبيعية أو ظواهر فيزيائية ما، ويكون محيطاً بالقوانين التي تحكمها، الأمر الذي لا يماثل في شيء تلك "المادة" الاصطناعية التي هي البرنامج. فالبرنامج هو عبارة عن "فكرة قابلة للتطبيق بشكل مباشر". وهذا أمر جديد تماماً. وتتطوي البرمجة على ممارسة الهندسة بهذا المعنى: أي استخراج واستخلاص وتجريد كيانات ما من العالم الحقيقي سنتمكن من تمثيلها والتحكم بها باستخدام وسائل آلية، ينبغي أن يماثل سلوكها سلوك كيانات العالم الحقيقي التي يحل البرنامج محلها.

من هذا المنظور، يمكن القول إن عملية البرمجة هي صنع أو تشييد رمز سيجد مكانه إما بين الأفراد، أو بين الأفراد والأنظمة، أو بين الأنظمة بعضها بعضاً. فهو، أي الرمز، يساهم في عملية تبادل المعلومات بين جميع كيانات المجتمع (من أفراد، ومنظمات وأنظمة)، ويخضع لجميع قيودها، وبخاصة فيما يتعلق بالتطور. إذ ينبغي، بقدر مساهمته في حياة المجتمع، أن يتطور بذات وتيرته؛ فتحن إذن بصدد لغة.

لقد عرّفنا علماء المنطق في حقبة العشرينات والثلاثينات من القرن العشرين، واللغويون إلى حد أقل، بمختلف مكونات نظام الرموز:

- المكون النحوي (والذي يسمى أيضاً بتركيب الجملة المنطقي)، والذي يقوم بمهمة تزويدنا بمجموعة، ربما تكون لا متناهية، من الصيغ المنطقية التي لا لبس فيها، والتي ستتيح توليد بنى متشابهة بين مختلف الحالات من خلال استخلاص المعنى. هو "نظام شكلي" * بمصطلح المنطق الرياضي، يعد بمثابة دعامة المعنى، لكنه ليس المعنى في حد ذاته.

- المكون الدلالي الخاص بدلالة البرنامج الذي يهتم بالصلة التي تربط المكونات المجردة المركبة بواسطة المكون النحوي، ومكونات العالم الحقيقي. وينبغي أن تتأى هذه العلاقة عن أوجه اللبس (ما يعرف بالثبات أو الاتساق المنطقي) وأن تكون مكتملة (أي تطابق كل مكون مجرد مع مكون من العالم الحقيقي، وكذلك بالعكس).

- المكون العملي الواقعي الذي يهتم بشروط فعالية "الرمز" من منظور المستخدم. ويعد سياق الاستخدام، والبيئة، والسلوكيات المتعلقة به، والجانب الجمالي وبيئة العمل، من عناصر هذا المكون.

من جهة أخرى نجد أن النظرية الرياضية الخاصة بالتواصل تستخدم نفس هذه المفردات، لتأكيد اهتمامها الحصري بترميز المكون النحوي للرمز. هذا وثمة علاقة وثيقة بين تصنيف البرامج ث وم وب وبين مفهوم هذه المكونات الثلاثة أي النحو والدلالة والواقعية. وإن الخلط ما بين هذه

* نظام شكلي: System Formel

المستويات الثلاثة مصدر لصعوبات بالغة التعقيد، لأن عمليات التصنيع الخاصة بكل من هذه المستويات مختلفة تماماً، تجدر الإشارة بشكل خاص إلى كون لغات البرمجة لا تعالج جيداً سوى المستوى النحوي، بيد أنه لا بد وأن تكون محايدة تماماً حيال المستوى الدلالي. ففي كل مرة حاولت فيها لغة برمجة فرض رؤيتها للعالم، كانت النهاية سيئة؛ ولعل نكسات لغة "أدا" الراهنة، وصعود نجم لغة سي، أبلغ شاهد على ذلك.

يمكن إذن اعتبار البرمجيات كنظام رموز ذات طبيعة لغوية أو لسانية. أما الدال، فنص البرنامج ذاته، وأما المدلول فهو الجزء الحقيقي الذي من المفترض أن يقوم البرنامج بنمذجته. فالبرنامج يحمل البنية، بيد أن المعنى أو القصد يبقى في العالم الحقيقي. فكما هو الشأن بالنسبة للغويات أو اللسانيات، فإن الحد الفاصل ما بين الدال والمدلول ليس واضحاً؛ في حين أن "الرمز" يتسم أيضاً بذلك الطابع الاعطباطي الذي لفت الانتباه إليه عالم اللسانيات، سوسور، فبمجرد ابتكاره، يخرج الرمز عن زمام سيطرة مبدعه. وتتوقف سمة البرنامج العملية أو الواقعية بشكل كبير على الخصائص النفسية الإدراكية التي يتصف بها المستخدم، وعلى ثقافته المعلوماتية، ومعرفته بنطاق عمل الرمز، وبالعلاقة الرمز برموز أخرى (المطور، والصائن، والدامج، ومسؤول الجودة الذين سيكون لكل منهم رأي مختلف، وفق وجهات نظر كل منهم).

وتكمن صعوبة البرمجيات الحقيقية في الجانبين المتعلقين بالدلالة والعملية، فهما المجالان اللذان ينبغي تخصيص الأهمية القصوى لهما. وينبغي أن تمكن الأساليب والأدوات المقترحة من قبل هندسة البرمجيات، مثل لغة النمذجة الموحدة (ULM)، النفاذ إلى هذه المكونات الأساسية، وألا يُكتفى بالتحكم بالبنية النحوية، ذات النطاق المحدود رغم أهميتها.

وقد تم تحليل جميع هذه التحديات على نحو مفصل في مؤلفنا قدرة وحدود الأنظمة المحوسبة (Puissance et limites des systèmes informatisés) (المرجع ١٨).

الفصل الرابع

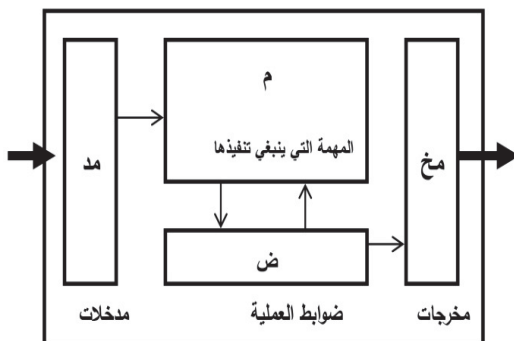
نموذج التطوير ودورة الحياة

١. مفهوم عملية الجودة

بدأت صناعة البرامج منذ حقبة سبعينات القرن العشرين كعملية متدرجة على مراحل (أول مقال لافيت بهذا الشأن كان مقال ديليو. ديليو. رويس عام ١٩٧٠ بعنوان: "إدارة تطوير أنظمة برمجية ضخمة"، جمعية مهندسي الكهرباء والإلكترونيات (IEEE) وسكون، أغسطس ١٩٧٠). وقد جرى العمل على توصيف ما يلي:

- كل من المراحل المتتالية لعملية "التصنيع"، منذ طلب الطرف المتعاقد للبرنامج وحتى لحظة الاستخدام.
- منطق تتابع مختلف المراحل.

بدأ في حقبة الثمانينات من القرن الماضي الاهتمام الجدي بالظروف التي ينبغي توفيرها لتعزيز السيطرة على جودة البرمجيات المصنعة، ولتسليم منتج متطابق مع المنتج المطلوب في الأساس. وفي سبيل ذلك، تم تفكيك عملية التطوير الإجمالية إلى عمليات أساسية وفق الرسم التالي:



هذا وتفكك العملية الأساسية إلى أربعة أنشطة تمثل نموذج (مد م ض مخ):

- تتطوي عملية "مد" على جمع المعايير التي ينبغي اجتماعها للبدء بالمهمة (مدخلات)؛
 - "م" هي المهمة التي ينبغي تنفيذها، فهي الغاية من عملية التطوير التي ستوكل إلى شخص أو فريق عمل أو عدة فرق عمل؛
 - "ض" تمثل الجزء الخاص بالتحكم بالعملية (ضوابط)، الذي يمكننا من خلاله التأكد من سير المهمة وفق قواعد الجودة السارية في المشروع كله، والخصة بمهمة ما؛
 - أما "مخ" فتجمع المعايير التي ينبغي تحقيقها لاعتبار المهمة منتهية (مخرجات).
- تربط الأنشطة المتعلقة بكل من "مد"، "ض" و "مخ" المهمة بمحيطها بشكل دقيق ومعتمد، بحيث تمكن معالجة أي حدث قد يؤثر على سير العمليات، وذلك بشكل واضح.
- ومن المثير للاهتمام هنا مطابقة النسبة المنجزة من المهمة، والتي تقدر بواسطة معايير محددة للعملية "مخ"، ونسبة الموارد "مد" المستهلكة والمقاسة بوحدة "أفراد-شهور". وتظهر هذه العلاقة التوافقية عادة على شكل منحنيات S يوافق نموذجها الرياضي المعادلة التفاضلية التالية:

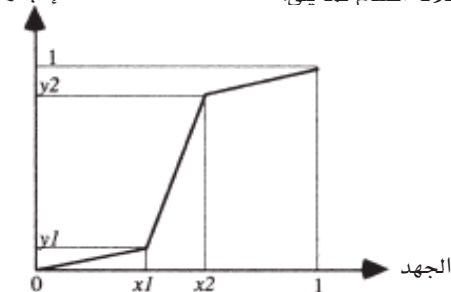
$$\frac{\Delta m}{\Delta t} = (\lambda - \exists) m$$

وبموجب هذه المعادلة فإن الزيادة النسبية Δm في إنجاز المهمة م بالنسبة إلى زيادة Δt مد في الموارد، تتناسب مع ما تم إنجازه ($\exists$) هو طرف رياضي من شأنه تسريع العملية بشكل أسي)، وما تبقى عمله (وهو طرف رياضي في المعادلة يبطئ العملية على نحو يتناسب مع ما تم إنجازه، ومن هنا كان الطرف التصحيحي (λm) . والحل التحليلي لهذه المعادلة هو

منحنى سيغموئيد (Sigmoide)، تركيبها الجبري كالتالي:

$$Y = \frac{be^{ax}}{1+ce^{ax}}$$

وباستخدام إحداثيات قياسية، وتبعاً للمنحني محل الاهتمام، نقوم بتمثيلها بواسطة ثلاثة أقسام كما يلي:



حيث:

$$1 > x_2 > x_1 > 0$$

$$1 > y_2 > y_1 > 0$$

والفترة:

$$[0: x_1] - \text{سرعة بدء التشغيل}$$

$$[x_1: x_2] - \text{سرعة التقدم}$$

$$[x_2: 1] - \text{سرعة الإنجاز}$$

وعادة ما تعتمد إما هذه الصيغة أو الأخرى، بما يلائم العملية

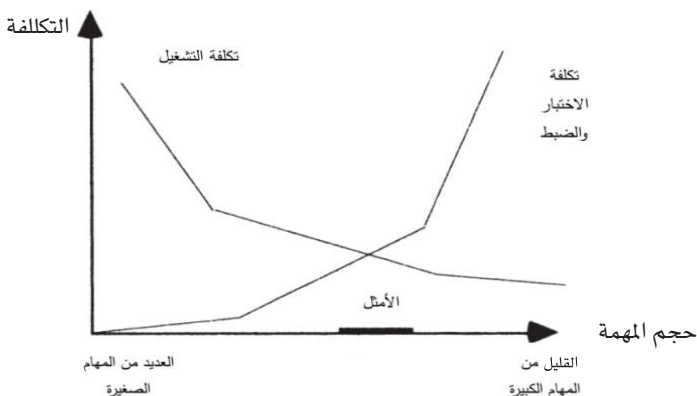
الحسابية.

يعكس هذا المنحني، وهو بالغ الأهمية لجميع الأعمال الهندسية، الطابع غير الخطي الخاص بعملية التطوير، وبخاصة المرحلة النهائية من المهمة، والتي قد تمثل فيها الـ ١٠٪ المتبقية ٣٠ إلى ٤٠٪ من الجهد

الذي لا يزال ينبغي بذله. ومن هنا تُرتكب أخطاء التخطيط الجسيمة حين يقتصر تفكيرنا على التفكير بأسلوب خطي.

تنطوي دورة التطوير المكتملة على مجموعة من العمليات البسيطة المقابلة للمهام المتنوعة اللازمة لإنتاج البرنامج. ويتوقف تسلسل العمليات على مدى تحقيق معياري مد المدخلات ومخ المخرجات.

جدير بالذكر أنه في عالم كامل لا يرتكب فيه الأفراد أي خطأ، تكون قيمة التكلفة الخاصة بالضوابط العملية ض تعادل الصفر، وتلك الخاصة بالمعيارين مد (مدخلات) و مخ (مخرجات) متدنية جداً. يبدو إذن عنصر ض فائضاً حقاً (في سياق نظرية المعلومات): فهو يعكس مواطن قصورنا الفردية والجماعية. وترتفع هذه التكلفة مع ارتفاع حجم المهمة المنفذة والحاجة لطلب المزيد من الضوابط. وإذا كان تفاعل البرنامج مع البيئة محدود النطاق، كانت الموارد المخصصة للضوابط محدودة، إلا أن عدد المهام سيكون أكبر بكثير، الأمر الذي سيتطلب مواردً بدوره (إذ نقوم بتضمين الضوابط داخل البرنامج دون تدخل من البيئة). ثمة إذن حالة موازنة بين الأمرين للوصول للأفضل كما هو موضح في الشكل:



من الواضح أن الأمثل يكون متغيراً، بيد أنه يتوقف على:

- كفاءة وخبرة الأفراد وفرق العمل؛

- الإدارة التي تقوم بتكوين فرق العمل.

في نمط العمل التسلسلي أي مخرجات عمل الفريق الأول تصب كمدخلات لعمل الفريق التالي، وفي حال اعتبرنا وجود عمليتين ١ و ٢، فإن إضفاء الطابع الرسمي على شرط الخروج من العملية ١ و شرط الدخول إلى العملية ٢ يقتضي صياغة عقد واضح بين مسؤولي المهام م ١ وم ٢. ومثل هذا الإجراء هو بمثابة شرط ضروري للوصول إلى الحالة الاقتصادية المثلى للتكاليف.

وفي حال لم يتحل المسؤول عن المهمة م ١ بالدقة، تسببت العملية ١ بتكاليف إضافية في العملية ٢، فما يكلف س للعملية ١ من شأنه أن يكلف العملية ٢ ك. س (حيث $K < 1$)، وهي العملية التي لا بد وأن تكون أقل قدرة من ١ على تصحيح الأخطاء التي لا تعزى إليها.

٢- دورة الحياة الاسمية ومختلف مراحلها

تتطابق المصطلحات المعتمدة هنا فيما يخص نموذج دورة حياة المنتج البرمجي مع تلك الخاصة بنموذج الميعار المذكور في المراجع [٤]. أي يتكون هذا النموذج من ٧ مراحل أو أوجه نشاط رئيسية سنقوم بتفصيل كل منها بإيجاز.



١ - **المرحلة الأولى**؛ أهداف البرنامج الحاسوبي: تقدم هذه المرحلة الأولى من تطوير أي مشروع برنامج حاسوبي وصفاً وتحليلاً شاملاً للمتطلبات التي ينبغي أن يلبيها البرنامج الحاسوبي. وهي إلى حد ما بمثابة حدود المخطط الذي نفكر فيه في سياق ما يلي:

- السياق الاقتصادي (تحليل قيمة الخدمة المقدمة، وتقدير الموارد التي يمكن حشدتها كمّاً ونوعاً للمشروع، والمواعيد النهائية المرغوبة، والعوائد المتوقعة على الاستثمار) وسياق الاستراتيجية الفنية (جدوى المشروع بصفة عامة، وخطة الجودة، والاستحواذ، واختيار مواد وتقنيات التطوير: لغات البرمجة، ونظام إدارة قواعد البيانات، والشبكات، وأدوات هندسة البرمجيات بمساعدة الحاسوب)؛

- سياق المخاطر، لاسيما في حال المشاريع الضخمة والبرمجيات التي تعد موثوقية عملها شرطاً أساسياً؛

- السياق التنافسي والميزات التنافسية، وبخاصة لدى تطوير طيف من الحزم البرمجية (ما يعرف بميزة المهاجم أو المبتكر)؛

- سياق التنظيم الشامل للمشروع، لاسيما علاقات السلطة المتعاقدة مع الشركة المطورة بالنسبة للمشاريع الكبيرة، ومشاريع الدولة ومشاريع تجميع أو إقامة ائتلافات تجارية، إلخ...

٢ - **المرحلة الثانية**؛ وصف المتطلبات: هذه هي المرحلة التي نقوم فيها بوصف الوظائف التي ينبغي للبرنامج الحاسوبي أن يقوم بها، وشروط التشغيل، وعقد الخدمة ومستوى الجودة المطلوب، دون الاكتراث (قدر المستطاع) بطريقة تنفيذ هذه الوظائف فعلياً. إذ أننا نعتبر البرنامج الحاسوبي بمثابة صندوق أسود لا نعرف سوى مدخلاته ومخرجاته، ونود

رغم ذلك أن نرى منه صفات محددة من أوجه الأداء (استهلاك الموارد، وسرعة الاستجابة، وتدفق المعلومات)، وموثوقية التشغيل (جاهزية النظام وأمنه).

كثيراً ما تُجرى هذه المرحلة، والتي تعد بالغة الأهمية بالنسبة للبرمجيات التي تكون فيها البيئة عاملاً غالباً (بـبرمج) بالتزامن مع المرحلة الأولى، لأن هاتين المرحلتين تخصصان في المقام الأول الطرف المتعاقد. هذا ويحدد توصيف المتطلبات الإطار التشغيلي العام الذي يتطور فيه البرنامج الحاسوبي ونسخه المتتالية. ويتم التحكيم والتسوية حال نشوب الخلافات، التي غالباً ما تحدث لدى تجاوز مشروع ما حجماً معيناً، استناداً إلى التحليلات التي تتم في مرحلة وصف المتطلبات.

٣- المرحلة الثالثة : التصميم: تهدف هذه المرحلة إلى تعريف وظائف وبناء البرمجيات على نحو بالغ الدقة، انطلاقاً من المتطلبات المحددة والقيود العامة المعينة خلال المرحلتين الأولى والثانية.

تسفر هذه المرحلة عن تحديد الاختيارات الفنية، وتعريف الوظائف (على نحو غير رسمي بلغة طبيعية وبلاستعانة بالرسوم الرمزية) ومن ثم على نحو رسمي بشكل أكبر باستخدام لغات البرمجة المخصصة مثل لغة تصميم البرنامج (بي دي إل)، وتتضح مجموعات الوحدات البرمجية، ويتم تحديد الخوارزميات التي ينبغي تنفيذها وتوصيفها بشكل كمي (معدل وقت الاستجابة، والموارد المستهلكة إلخ...)، فضلاً عن فهرسة البيانات الأساسية (التي تشترك فيها عدة وحدات أو تتبادلها مثل الرسائل) في قاموس، وتعيين بنية التحكم التي تُوصف تسلسل الوظائف، والأحداث التي تُطلق تسلسل العمليات بشكل كامل. كما تتم هنا فهرسة كل ما يتسم بالأهمية بعناية في مجموع مصطلحات برنامج الحاسوب (وهو ما يسمى بعناصر تكوين البرنامج الحاسوبي).

٤- المرحلة الرابعة: البرمجة واختبارات الوحدات: تنطوي هذه المرحلة على البرمجة الفعلية للوظائف استناداً إلى المعلومات الدقيقة الناجمة عن مرحلة التصميم. إذ تُترجم الوظائف إلى لغة أو لغات البرمجة التي تم اعتمادها، مع مراعاة معايير البرمجة المحددة في خطة ضمان الجودة. وفي هذه المرحلة من التطوير، تتحول النظريات والحسابات الفكرية والوصف الورقي إلى عناصر قابلة للتنفيذ ومن ثم يمكن التأكد منها عن طريق التجربة. تنتقل إذن من النظرية إلى تطبيق هذه النظرية وعندها تنكشف مواطن ضعفها. لا بد إذن، عند اكتشاف حالات إخفاق تعود إلى التصميم، من تحديث الوثائق الخاصة بالتصميم والبرامج ذات الأخطاء في آن واحد، تجنباً لمعضلات خطيرة في مرحلة التكامل ولدى ارتقاء البرنامج الحاسوبي لاحقاً.

وبنهاية مرحلة البرمجة، فإن كتل البرنامج الحاسوبي، أو ”وحدات الترجمة“، والتي تجمع عدة وظائف، ينبغي أن تترجم إلى لغة الآلة دون أخطاء وأن تنفذ بالإجراءات الاختبارية التي تضمن سلامة منطق البرنامج (أي الحصول على نتائج سليمة ضمن المهل الزمنية المحددة وباستهلاك الموارد المتاحة).

٥- المرحلة الخامسة: تكامل المكونات واختبارات التأهل: يتم في هذه المرحلة تجميع جميع كتل البرنامج أو مكوناته تدريجياً بشكل يضمن التحقق من سلامة عمل البرنامج الحاسوبي والمصادقة عليه حتى يمكن تشغيله في بيئته الحقيقية.

عند نهاية مرحلة دمج وتكامل المكونات، نستطيع لدى تشغيل البرنامج واتباع الإجراءات الملائمة، التأكد من أنه يفي بجميع الأهداف الوظيفية المحددة في المرحلة الثانية.

٦- المرحلة السادسة: التثبيت أو التركيب: هي المرحلة الخاصة بتشغيل البرنامج الحاسوبي في بيئة العمل. تنطوي إذن هذه المرحلة على التأكد من سير عملية التثبيت وفق متطلبات فريق التشغيل لدى العميل، ومن وجود وسائل التدريب، ومن دخول الدعم الفني للمورّد قيد العمل.

٧- المرحلة السابعة: التشغيل والصيانة: هي المرحلة الخاصة بوضع البرنامج الحاسوبي قيد استخدام جميع المستخدمين المتوقعين بادئ الأمر، وذلك دون قيود. وتهدف هذه المرحلة، التي قد تطول كثيراً لدى بعض أنظمة برمجيات التسليم الجاهز أو المفتاح باليد (إدارة الإنتاج، مراقبة الطيران، الاتصالات الهاتفية، إلخ...)، إلى ضمان عمل البرنامج الحاسوبي الذي تم تثبيته على النحو السليم؛ من هنا لا بد من تصحيح أوجه الخلل والأخطاء التي تظهر في مرحلة التشغيل ضمن المواعيد الزمنية المحددة في عقد الصيانة المبرم بين المشغل ومحرر البرنامج.

٣ - دورة الحياة على شكل V للنظم البرمجية التي من حجم معين

المبرمج هو ذلك «المتفاعل» الذي يحول المعلومات إلى برامج قابلة للتنفيذ على الحاسوب. وهذا تقيض ما يمكن ملاحظته في قطاعات هندسية أخرى يسهل فيها استبدال الإنسان بإنسان آلي. وباستطاعة مبرمج موهوب انتاج برنامج ينطوي على ٢٠٠٠٠ من التعليمات التي تم اختبارها وتوثيقها خلال عام واحد (ما يعادل اصدار كتابين كل منهما ٤٠٠ صفحة بمعايير النشر)، بيد أن الوسطي الشائع للمبرمج العادي هو حوالي ٥٠٠ من التعليمات.

ولتعزيز هذا المعدل وتقليص الوقت المستهلك، يجري العمل في فرق مكونة من عدة أفراد يتقاسمون المهام وفق مهاراتهم وخبراتهم الخاصة، علماً بأن رئيس فريق ما يكون عادة الفرد صاحب الخبرة الأوفر. وتتوقف فعالية أداء الفريق في المقام الأول على قدرته على التواصل، وعلى مدى استعداد عناصر هذا الفريق للتعاون. ففي حال كان ثمة عدد ع من الأفراد، فإن عدد النقاشات التي ينبغي إجراؤها للتوصل لاتفاق (وهو شرط أساسي لتحقيق التنسيق في العمل) يعادل $\frac{E(E-1)}{2}$ الأمر الذي يحد بالضرورة من عدد أعضاء فريق العمل. وعلى الصعيد العملي، نلاحظ اعتماد معدل ٥ إلى ٧ أشخاص في فريق العمل، بما في ذلك رئيس الفريق. أما إذا زاد العدد عن ذلك، فلا بد من قدرة على التواصل وسلطة استثنائية لضمان تناسق فريق العمل.

وعندما يتجاوز حجم المشروع حداً ما، ينبغي تجزئة مشروع البرنامج الحاسوبي إلى نظم فرعية يتناسب حجم تطويرها مع مدى ما يستطيع الفرد أو الفريق أو عدة فرق، توليه ومعالجته.

عملية تجزئة مشكلة «كبيرة» إلى عدة مشكلات فرعية أصغر حجماً هي إحدى نتائج مرحلة التصميم البالغة الأهمية، وهي المرحلة التي بدورها تُجزأ إلى عدة مراحل فرعية وفق مستويات التجريد التي من

شأنها فصل وترتيب علاقات الاعتماد بين المشكلات الفرعية حسب الأولوية. وهذه عملية تتطلب خبرة مديدة.

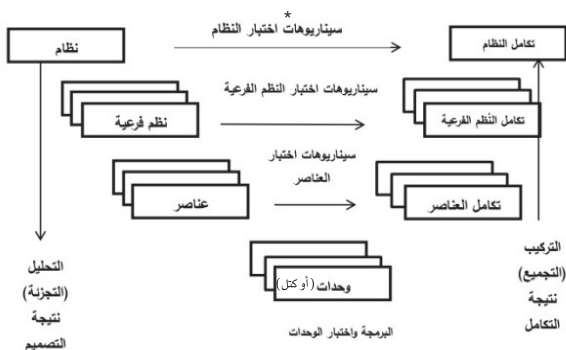
على وجه الخصوص، ينبغي النظر إلى:

- تصميم النظام على نحو يراعي أوجه تبعية البرنامج الحاسوبي لمحيطه والتي لا مفر منها. كما يجري تقسيم النظام البرمجي إلى عدة أنظمة فرعية ينبغي أن تتمتع بأقصى قدر ممكن من الاستقلالية؛

- تصميم النظم الفرعية مع تجزئة النظام الفرعي الواحد إلى عناصر أصغر حجماً حينما يفوق حجم النظام الفرعي قدرة فريق عمل ما؛
- تصميم العناصر أو التصميم التفصيلي الذي ينطوي على تحديد عمل فريق البرمجة. تتم هنا تجزئة العنصر إلى وحدات أو كتل يكلف بها المبرمج.

يقوم مبدأ التجزئة على أساس مجموعات الوظائف التي تشترك في خاصية أو أكثر، ما يجعل تجميعها أمراً مثيراً للاهتمام (جمع بعض وظائف الخدمة المتماثلة مع بعضها)، أو التي تضمن تكامل الخدمة.

تسفر هذه العملية عن تجزئة المشكلة إلى وحدات أو كتل قابلة للبرمجة، ينبغي بعد ذلك تجميعها وفق ترتيب معين لتشييد النظام الحاسوبي فعلياً. ولا يكون هذا ممكناً إلا بمراعاة بعض قواعد اللعبة الخاصة بكل نظام فرعي وبالنظام ككل. وتشكل هذه القواعد، التي تحدد بروتوكولات تواصل العناصر فيما بينها، الواجهات البينية أو الواجهات. يقابل هرمية مستويات التجريد إذن هرمية من الواجهات التي ينبغي أن تكون شديدة الترابط المنطقي فيما بينها، كما هو شأن مختلف المسلمات المؤلفة لنظرية رياضية ما. فهذا هو شرط إتمام عملية التجميع أو التركيب، التي تسمح بإجراء التكامل، وعمليات الفحص والتحقق الموافقة. ويرمز إلى هذه العملية بمجملها بـ "دورة الحياة على شكل V" (أنظر الشكل التالي).



٤. عمليات التطوير غير المكتملة :

«الماكيت» والنماذج البدئية

كما هي الحال في مجالات أخرى من الهندسة، لا يكون أحياناً ثمة خيار سوى إجراء أو تنفيذ بعض الوظائف للتمكن من تحديدها على النحو الصحيح؛ وهذا أكثر ما يكون صحيحاً بالنسبة للبرامج التي من النوع ب.

١. الماكيت (نماذج المحاكاة) - لا شك في أن صنع نماذج المحاكاة أو

الماكيت مفيد في تسهيل التعبير عن المتطلبات، وتحديد مواصفات النظام البرمجي التي سيحتك وإياها المستخدم بشكل مباشر؛ وهو ما نسميه بواجهة المستخدم (أو الواجهة بين الإنسان والآلة). وقد بات هذا النوع من الواجهات البينية ذا أهمية مع انتشار الشاشات المرئية عالية الوضوح.

٢. النماذج التجريبية : يخدم النموذج التجريبي أغراضاً متفاوتة،

إذ أننا نسعى من خلاله لبناء نظام غير مكتمل، إلا أنه حقيقي من حيث الأبعاد بحيث نتمكن من إجراء اختبارات بمقاييس حقيقية. ويمكن هنا اتباع أحد نهجين:

* يمكن استعمال مصطلح (مشاهد) بدل (سيناريوهات):

أ- نستطيع أن نحدد مجموعة فرعية صغيرة من وظائف النظام غير المكتملة، بحيث نقوم بإجراء أول زيادة على البرنامج الحاسوبي. ويجب أن ينطوي هذا النموذج التجريبي على أقصى حد من التعليمات الخاصة بقياسات البرمجيات في النظام، وذلك بهدف تقويم أداء هذا النظام البرمجي.

ب- بإمكاننا تغيير لغة البرمجة الخاصة ببعض مكونات النظام، أو تبني وسائل برمجية خاصة (على سبيل المثال «التفسير»*) أي استعمال لغة قابلة للتنفيذ من خلال الآلة مباشرة بدلاً من استعمال الترجمة (إلى لغة الآلة)، أو استبدال بعض العناصر ببرامج جزئية لا تستطيع الاستجابة سوى لحوافز محددة، علماً بأن الهدف هنا أيضاً هو تشييد نظم فرعية تمثل النظام بشكل أسرع، وإن كان ذلك مقابل التضحية إلى حد ما ببعض جوانب الأداء.

هذا النوع من التطوير هو واسع الانتشار، إن لم يكن أساسياً لدى تطوير النظم البرمجية الضخمة التي يتم صنعها دائماً بأسلوب الزيادات المتتالية على البرنامج.

٥. الصيانة والارتقاء

تبدأ الصيانة ببدء المرحلة السابعة، لدى القيام بتشغيل النظام البرمجي. وتختلف الصيانة جذرياً وفق ما إذا كان نظاماً «جاهزاً للاستخدام الخاص» أو «حزمة من البرمجيات» التي يتم توزيع ملايين النسخ منها. وتنطوي الصيانة على طيف واسع من الأنشطة التي تتراوح بين تصحيح الأخطاء المعروفة، وهو ما نسميه صيانة الإصلاح، وحتى إضافة التعديلات المتفاوتة الحجم بناء على طلب المستخدمين. ويكون

* التفسير: interpreter وهي الكتابة بلغة تنفذ مباشرة (دون المرور بمرحلة الترجمة إلى لغة الآلة أي دون complete).

الحديث حينئذ عن ارتقاء النظام البرمجي؛ علماً بأن الصيانة الارتقائية شديدة الشبه بعملية التطوير.

١ - الصيانة الإصلاحية :

أ) الأنظمة الجاهزة للاستخدام (برمجيات التسليم الجاهز أو المفتاح باليد) : تنطوي هذه الصيانة على إصلاح جميع الأخطاء التي يتم الكشف عنها طيلة مدة استخدام النظام، وقد تكون هذه المدة طويلة جداً تصل إلى ١٠ أعوام بل حتى ٢٠ عاماً بالنسبة لبعض النظم البرمجية بالغة الاندماج في معدات ضخمة: كأجهزة التحكم بالطيران، والأجهزة النووية، وأجهزة التحكم بإشارات السكك الحديدية، ونظم الأسلحة والخ...

تختلف هنا عملية الصيانة كلياً عن عملية التطوير، ولكنها ترث عن عملية تطوير ما يلي:

- يكون فيها إطار بنيان النظام محدداً بشكل تام؛
- تكون أدوات ووسائل التطوير موضوعة ومستخدمة بشكل مكثف، فهي معروفة تماماً إذن.

وهي عملية لا تنطوي على الإبداع مثل عملية تطوير البرامج، بيد أنها تتطلب قدراً بالغاً من التنظيم والانضباط لا سيما وأن حجم المعلومات التي يقوم مسؤول الصيانة بمعالجتها كبير.

ولا يكون المسؤولون عن الصيانة عادة هم مؤلفو البرامج المسؤولين عن تغييره. وفضلاً عن فهم البرامج المراد صيانتها، سيكون على مسؤول الصيانة استخدام وثائق النظام البرمجي والسجلات التاريخية والاختبارات التي أجريت. فجودة توثيق النظام البرمجي، واتساقه مع هذا النظام، يعد عاملاً حيوياً لفعالية أداء العاملين في الصيانة. إذ يقوم مسؤول الصيانة بالتعامل مع عمل عدة مبرمجين، لذا يفضل أن يكون أسلوب التوثيق والبرمجة متجانساً قدر المستطاع، الأمر الذي يتطلب اتخاذ معايير صارمة فيما يتعلق بالتوثيق والبرمجة. ذلك أن الإفراط في التساهل في مراحل التصميم والبرمجة سيسفر لا محالة عن تكاليف

باهظة في الصيانة ربما تقوض كلياً ربحية المشروع. الانتقال إذن من التطوير إلى الصيانة مرحلة بالغة الدقة، وبخاصة أن الأمر يتعلق في كثير من الأحيان بعدة منظمات، بل عدة شركات مختلفة، ربما لا تتعاون مع بعضها البعض على النحو السليم. وعندما يكون ثمة قدر مفرط من الأخطاء، يقول فريق الصيانة إن فريق التطوير لم يؤد عمله كما يجب، بينما يجد بعض المبرمجين أن من ”الطبيعي“ أن تُترك مهمة الصقل النهائي لفريق الصيانة: لا مفر حينئذ من الخلاف.

(ب) حزم البرمجيات: تنطوي صيانة حزمة البرمجيات على جميع عناصر النظم البرمجية الجاهزة للاستخدام المذكورة اعلاه مع وجود تحدٍ إضافي، هو الأخذ بالاعتبار مجموعة الأنظمة التي ستثبت فيها حزمة البرمجيات أو ما يسمى «بمنصة العمل*»، وهو ما يُسمى تأثير المنصة. إن خطأ واحداً في حزمة البرمجيات من شأنه عادة أن يولد أخطاء ومواطن خلل ربما تظهر بشكل مختلف حسب سياق استخدام النظام البرمجي. ويُترجم هذا على الصعيد العملي، إلى تدفق تقارير الأخطاء، التي يرسلها مسؤول الاستخدام الموجود في موقع العميل إلى خدمة الصيانة لدى محرر حزمة البرمجيات. وتتوقف كثافة تدفق هذه التقارير (عدد تقارير الأخطاء للوحدة الزمنية الذي يجب معالجته وفق وتيرة استخدام جهاز الحاسوب وحجمه) على عاملين أساسيين:

- عدد الأخطاء المتبقية في حزمة البرمجيات بعد مرحلة تجميع (تكامل) العناصر والاختبار (المرحلة الخامسة). ولا يكون هذا الرقم معروفاً إلا أنه يمكن تقديره [٥]؛
- عدد الأنظمة التي تشكل ”المنصة“ الخاصة بحزمة البرمجيات، إذ تعدد أوجه استخدام الحزمة البرمجية (استخدام مستمر، متقطع، استثنائي...) تبعاً لاستعمالها في أجهزة متفاوتة القدرة.

* (منصة العمل) هو مصطلح parc بالفرنسية و Platform بالإنجليزي:

ويتم التعبير عن معدل أو وتيرة الاستخدام بعدد ساعات الاستخدام معياراً بقدرة المعدات.

ولاستيعاب تأثير «المنصة»، ينبغي النظر إلى حالتي حدوث الأخطاء:
- الحالة العابرة التي تحدث عند عملية التثبيت الأولى أو عند استبدال نسخة من حزمة البرمجيات بنسخة لاحقة. وهذه الحالة هي المرحلة السادسة من دورة الحياة والتي تتصف بمواطن اضطراب مرتبطة باتساع المنصة على نحو سريع؛

- الحالة المستقرة التي تميز المرحلة السابعة، والتي توافق إدارة منصة مستقرة أو بطيئة النمو.
فلنقم أولاً بوصف الحالة المستقرة.

في تاريخ ت تتميز المنصة بعدد من الأنظمة ن_١، ن_٢، ن_٣، ...، ن_٦، ن_ض، ومجموعة من الحالات الفنية لحزمة البرمجيات ح_١، ح_٢، ح_ر، ح_د. وتكون هذه المصفوفة بالغة التعقيد بالنسبة لحزم البرمجيات واسعة الانتشار وهي عادة (قاعدة بيانات).
يتم الإعلان عن خطأ ما في النظام ن_١ الذي يقوم بإرسال تقرير أخطاء؛ ويكون ن_١ في حالة ح_ر. يُرسل تقرير الأخطاء للشخص المناسب (ليس الأمر سهلاً بالضرورة!) الذي يباشر بعملية الإصلاح. ربما تحدث عند ذلك إحدى الحالتين التاليتين:

- يكون الخطأ معروفاً من قبل، ويكون الإصلاح قد نُفذ في نظم أخرى، فيتم إجراء الإصلاح المطلوب مباشرة في ن_١ (فترة ٤٨ ساعة على الأغلب).

- يكون الخطأ مستجداً، وينبغي إيجاد أسلوب لإصلاح ن_١. ولدى إيجاد وسيلة التصحيح يجري تنفيذها على النظام ن_١ (فترة ٧٢ ساعة على الأقل)، ثم بعد فترة متابعة يجري الإصلاح على نظم أخرى ذات تركيب مشابه يجعلها قابلة لاحتواء نفس الخطأ، ومن ثم أخيراً على

مجمل المنصة (قد يمتد انتشار الخطأ على مدار عدة أسابيع).
 بذلك تكون الحالة الفنية للنظام قد انتقلت من ط إلى ط+١.
 يتضح إذن أنه في اللحظة ت، تجتمع عدة حالات فنية في المنصة،
 وتكون بنية هذه المجموعة هي محل اهتمام إدارة التركيب. ويتوقف تنوع
 هذه المجموعة على عدد الأخطاء المتبقية، وعلى سياسة توزيع الإصلاح
 حيث تدخل عوامل تجارية (طبيعة عقود الصيانة، وحجم الخلل، إلخ...).
 عند اللحظة ت، يكون لدينا إذن ع_١ نظاماً في الحالة ح_١، وع_٢ في
 الحالة ح_٢،....، ونظاماً في حالة ح_{١+ط}. ويكون ن_٣ قد انتقل من حالة ح_١
 إلى الحالة ح_{١+ط}.

تتم كل عملية تثبيت جديدة وفق الحالة الفنية التي تُعد الأنسب، أو
 الأقل خطراً في سياق استخدام حزمة البرمجيات، وهي ليست الأحدث
 بالضرورة.

ربما يحملنا الحكم السريع على الاعتقاد بأن الوضع الأفضل يكون
 ذلك الذي لا توجد فيه سوى حالة فنية واحدة في المنصة. بيد أن الأمر
 ليس كذلك على الصعيد العملي، فالأمر الذي يرجح قرار إضافة إصلاح
 هو دالة التكلفة المرتبطة بحالة المنصة. ويمكن رسم هذه الدالة على النحو
 التالي:

- يميل عدد الحالات الفنية إلى الزيادة مع تقدم عمر المنصة.
- تميل تكلفة إدارة المنصة إلى التنامي، لأن ثمة عدد متزايد من
 العناصر التي ينبغي إدارتها وطيف متنوع متزايد من الحالات (علماً
 بأن أسوأ الحالات هي التي يكون فيها لكل نظام حالة فنية خاصة
 به).

إن أردنا خفض عدد الحالات الفنية، يجب توزيع أوجه الإصلاح بشكل
 منهجي على كل المنصة، وهو أمر مكلف، وينطوي على مخاطرة (فقد تختل
 بعض المواقع وتكشف عن مواطن خلل أخرى)، بل إنه قد يكون غير ذي

فائدة لأن الأخطاء التي تظهر في آن واحد في جميع المواقع نادرة جداً (وفي هذه الحالة تنم عن إخفاق المرحلة الخامسة من دورة الحياة الإسمية V، وهذه في حد ذاتها مشكلة أخرى). وحين تكون المنصة كبيرة، قد تكون تكلفة تحديثها باهظة جداً، وبكل الأحوال، لا يمكن معالجة كل مواقع المنصة في آن واحد.

من هنا فإن ما بين الوضع المثالي لعالم كامل لا خطأ فيه، وبين حالة الفوضى القصوى، ثمة حال اقتصادي أمثل يتحقق بعدد معين من الحالات الفنية التي ستتيح معالجة مواطن الخلل الجديدة، التي يتم الكشف عنها، من خلال تطبيق إما حالة فنية معروفة أو جديدة، بشكل فوري أو على عدد من المواقع أو المقرات المتشابهة.

ويقوم فريق الصيانة بإدارة الحالة المستقرة دون اللجوء إلى فريق التطوير سوى في حالات استثنائية.

تتميز الحالة العابرة بمواطن عدم اليقين إزاء سلوك حزمة البرمجيات ومحيطها لذلك:

- ينبغي تثبيت حزمة البرمجيات تدريجياً، وهو ما يسمى بالتسليم المضبوط المراقب، بحيث يتم التأكد من سلوك حزمة البرمجيات لدى الاستخدام الحقيقي، وذلك بهدف التمكن من الاستجابة بشكل فوري حال ظهور مشاكل، إما مع فريق الصيانة و/أو الدعم الفني، أو مع فريق التطوير مباشرة إن لم يكن فريق الصيانة قد دخل مرحلة التشغيل؛

- ينبغي إدخال فريق الصيانة إلى الحيز التشغيلي تدريجياً حتى يتأقلم مع المنظومة، علماً بأن منحى التعلم هو منحى أسّي تقليدي (على شكل S).

- ينبغي أيضاً إدخال فريق تشغيل النظام إلى الحيز التشغيلي تدريجياً حتى يتعلم استخدامه، ما يعني بالنسبة لمصمم البرمجيات المزيد من تقارير الأخطاء.

يؤول التسليم المتسرع الذي يتم في غير أوانه إلى إنهاك فريق التطوير لا محالة، وزعزعة مسؤوليات فريق الصيانة/الدعم الفني، فضلاً عن رسم صورة تجارية سيئة ربما تقوض بكل بساطة عملية البيع (وهي الحالة التي يستغلها المنافسون طبعاً في السوق التي تستخدم فيها المنافسة). المرحلة السادسة إذن مرحلة تتطوي على مخاطرة عالية تتطلب إعداداً دقيقاً في نهاية المرحلة الخامسة. وثمة اتفاقيات بين كبار ناشري حزم البرمجيات وأكثر عملائهم ولاءً لتجربة النسخ المستحدثة من منتجاتهم تجريبياً أولاً بشكل آمن قبل الدخول في المرحلة السادسة.

٢- ارتقاء البرمجيات:

تصاحب عملية تشغيل حزمة البرمجيات عادة طلبات إجراء تعديلات من قبل مستخدمي هذه الحزمة. وينبغي النظر إلى هذه الطلبات على أنها علامة إيجابية تدل على مدى تفاعلية المنصة. ويمكن تصنيف هذه الطلبات في مجموعتين:

- تلك التي ينبغي اعتبارها أخطاء ومن ثم التعامل معها على هذا الأساس (تعديل في العوامل الإنسانية في بيئة العمل أو في تعامل الانسان مع الآلة، في سهولة الاستخدام، في أوجه الأداء، إلخ...). يكون الحديث حينئذ عن الصيانة التكميلية:

- تلك التي ينبغي اعتبارها إضافات لتلبية متطلبات جديدة، يكون في مصلحة ناشر الحزم البرمجية أي الشركة المطورة للبرمجيات إضافتها لأية نسخة لاحقة من منتجه.

- يكون حديثنا في الحالة الثانية عن الارتقاء، لأننا نقوم فعلياً بتعزيز المنتج بالنسبة لبيئته من خلال تلبية متطلبات المستخدم على نحو أفضل. عملية الارتقاء هذه هي جانب آخر من التطوير والذي يتسم بكل مما يلي:

- صيانة حصرية لأن ثمة إرث ينبغي إدارته، وينبغي أن تتم إضافة

الوظائف الجديدة دون تكلفة إضافية على الأنظمة المثبتة.

- عملية تطوير خالصة بكل خصائصها لأن حجم الإضافات ربما يكون كبيراً (مثل تكييف حزمة البرمجيات مثلاً لبيئة مادية جديدة).
على الصعيد العملي، تتناوب -عقب مرحلة التصميم المبدئية- دورات تهيمن عليها بشكل متناوب الصيانة (الإصلاح والتكييف) وعمليات الارتقاء. وفي حالة حزمة البرمجيات، يتم تخصيص فترة توقف في عملية التطوير، لضمان ربحية المنتج على الأقل واختبار ردود فعل المنصة قبل الانطلاق في دورة أخرى من الارتقاء (مع فريق تطوير جديد).
يشكل مجموع تكاليف مختلف هذه النسخ إضافة إلى تكاليف صيانة الإصلاح، الخاصة بحزمة البرمجيات، تكلفتها الكاملة. وإذا ما أخذنا بعين الاعتبار تكاليف التطوير والصيانة الارتقائية فقط، يمكن التوصل إلى نسب التكلفة وفق المراحل حسب النسب المئوية التالية:

النسخ الثانية والثالثة		النسخة الأولى	
التصميم	٢٠	التصميم	٤٠
البرمجة	٢٠	البرمجة	٢٠
التجميع أو التكامل	٦٠	التجميع أو التكامل	٤٠

نلاحظ أنه إذا طال عمر حزمة البرمجيات وارتفع عدد نسخها المتتالية، وجب الحرص بشكل خاص على مرحلة التجميع والتكامل، أي كل ما يتعلق بتخطيط وتنفيذ الاختبارات وهو ضرورة أيضاً لصيانة إصلاح جيدة.

بهذا كله تكون عملية الصيانة عملية بالغة التعقيد من حيث التنظيم والتخطيط والتنفيذ. ومن أهداف هندسة البرمجيات إيجاد سبل وأدوات لتعزيز إدارة هذه العملية المعقدة، وبخاصة الاختبارات.

٦. الهندسة العكسية للبرمجيات

يتنامى معدل تكلفة التعديلات التي تُجرى على حزمة البرمجيات طيلة فترة الصيانة بتقدم عمر حزمة البرمجيات تنامياً كبيراً نظراً للأسباب التالية:

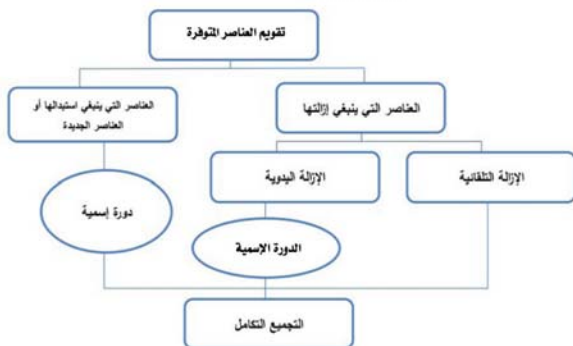
- يتم استنفاد (إشباع) البنية المعمارية والواجهات التي تدعم النظام البرمجي ولا يعود ممكناً القيام بأية إضافة لها دون التسبب في تراجع وظيفي مكلف في المنصة المثبتة.
- يتهاوى التوثيق الفني.
- تفقد الاختبارات فعاليتها.
- تكون فرق التطوير والصيانة قد تجددت وتغيرت عدة مرات، الأمر الذي يسفر عن تبديد المعرفة.
- يعود تدفق الأخطاء، بعد فترة تحسن مستمرة واستقرار، إلى التنامي، بل وإلى التآرجح، وهي ظاهرة أكثر إثارة للقلق، الأمر الذي يعني أن ثمة أخطاء جديدة يتم إضافتها، وأن هذه الأخطاء تنتقل حسب مستوى إشباع الواجهات.
- عند الوصول لحالة من هذا النوع، ثمة سلوكان ممكنان:
- إعادة صنع كل شيء، وطلي صفحة الماضي. وقد كان هذا هو السلوك السائد حتى حقبة الثمانينات.
- محاولة استعادة جزء مما سبق صنعه، بتحديثه، وهو الاتجاه الواضح في حقبة التسعينات.
- وتُعرف عملية الاستعادة أو إعادة التدوير هذه بالهندسة العكسية لحزمة البرمجيات.

في حزمة البرمجيات الكبيرة، التي تتطوي على مزيج من برامج من نوع ث وم وب، تكون العديد من الوظائف قابلة للاستعادة. وبالنسبة للنظام البرمجي الجديد، يتحول كل ما كان من نوع م أو ب إلى ث، لأننا،

كما هو جلي، نتعامل مع برنامج معروف المواصفات تماماً. وتكون تكلفة التطوير أقل.

جدير بالذكر أن العناصر التي تسهل استعادتها هي:

- التوثيق الخاص ببيان النظام.
- شفرة المصدر.
- البيانات في حالة النظام البرمجي الذي يستخدم ملفات أو قواعد بيانات
- الاختبارات.
- بعض الأدوات المطورة خصيصاً (مثل محرر البيانات الخاصة بقياس أداء بعض أجزاء النظام، ومراقب الاختبارات)، وإلى حد أقل، بعض الوظائف التحليلية أو الاستطلاعية السابقة للتعبير عن المتطلبات.
- في حال انتهاج سياسة إعادة الاستخدام، ينبغي أرشفة جميع هذه المكونات بحرص وعلى نحو يسهل النفاذ إليها (هنا أيضاً لا يكون الأمر سهلاً وواضحاً ويتطلب تخطيط وتنظيماً متقناً).
- ثمة تعديل عميق يصيب دورة التطوير الإسمية V لدى تنفيذ هذه السياسة كما يبينه الشكل التالي:



$$م = ٠,٩٩ = ع = ١٠٠ = م = ٠,٣٧$$

$$م = ٠,٩٩٩ = ع = ١٠٠٠ = م = ٠,٣٧$$

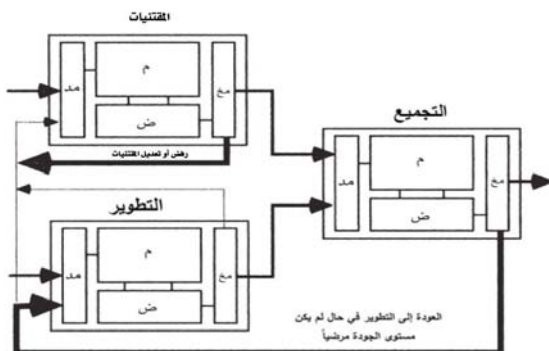
وهذا يفرض معضلة جدية إزاء طول السلسلة (حالة حزم البرمجيات التي لا تتوقف).

ينطوي إذن تجميع أو تحقيق تكامل نظام كبير، على عملية تجميع عناصر ما، بغرض الحصول بشكل منهجي، وتلقائي قدر المستطاع، على مجاميع المكونات التي تم التحقق من سلامة عملها والمصادقة عليها، متنامية وقريبة بشكل متزايد من النظام النهائي. ويتبع مجرى هذه العملية منحى أسي تقليدي على شكل S.

يتضح هنا إذا أنه في حال كان بيان النظام غير سليم (اتسمت على سبيل المثال المكونات بتبعية مترابطة على نحو مفرط)، وأغفلت عملية التخطيط قياس أثر المنحنيات الأسية، فإن تكلفة، بل ومدة مرحلة التجميع، ربما تعرض للخطر مشروعا، ربما يظهر على الورق، كل مظاهر المشروع الجيد.

مرحلة التجميع هي أيضاً الحيز الذي نقوم فيه بتجميع ما تم تطويره خصيصاً للمشروع، وما تم اقتناؤه (المواد + نظم التشغيل + الحزم البرمجية). وتشير التجارب إلى أن عمليات الاقتناء هذه لا تتفق دائماً مع "دفتر شروط" المطور/المحرر، الذي يميل بشكل طبيعي إلى الإفراط في الطلب، وبخاصة فيما يتعلق بالأداء وأمان الأداء. لذا فإن من الضرورة بمكان التأكد سابقاً من طلبات الاقتناء، قبل إدراجها في عملية التجميع، لأنها إن لم تكن مرضية فلا حل إلا تغيير المزود، وهي عملية بطيئة جداً. وينبغي النظر بمنتهى الدقة في ضم هذه العمليات المختلفة- تطوير، وتبديل وتجميع النظام- لأنها متباينة الوتيرة.

باستخدام نموذج مدخلات، مهام، ضوابط، مخرجات (مد م ض مخ) يمكن رسم مبدأ الضم كما يلي:



٨. أدوات هندسة البرمجيات و«ورشها»

١ - أمور عامة :

إن الوظيفة أو العمل المطلوب يخلق الأداة اللازمة التي بدورها تؤثر على هذه الوظيفة. وهذا ما يُطلق عادة عملية توفيق دوائر تسفر، بعد التكرار عدة مرات، عن أفضل حالة من الناحية الاقتصادية تُوفِّق ما بين صعوبات التحدي الذي ينبغي معالجته، وكفاءة المتصرف، والاحتمالات التي تتطوي عليها التقنية. وهذه القاعدة العامة تنطبق بالطبع على عملية تصنيع النظام البرمجي.

إن الظروف التي تفرض وجود أداة هي:

- تكرار النشاط و/أو المهمة، لأن الآلة لا تتعب.
- سعة أو كبر حجم النشاط أو المهمة لأن الآلة أعظم قدرة من الإنسان على أداء مهام معينة.
- الدقة العالية و/أو التحقق من سلامة العمل، فالآلة تعمل بولاء ودون ارتكاب أي خطأ وفق خطة معدة مسبقاً.
- وعلى النقيض من ذلك، فإن عمالاً إبداعياً متفرداً بطبيعته، يتطلب قدرة الإنسان على المعالجة الرمزية، وثقافته وتجاربه وتفكيره

السليم، هوغير قابل عمومًا للتحويل إلى أداة، أو للأتمتة. أما نتيجة هذا العمل، فيمكن التعبير عنها بلغة ومجموعة من الرموز المخصصة (ad hoc). أن تميز هذا الاختلاف بين العاملين المذكورين، أي الذي يحتاج لأداة والذي لا يحتاج بالغ الأهمية، إذ أنه لا بد من تمييز نظام الترميز، عن الشيء الذي يقوم بوصفه.

كما ينبغي إدراك حقيقة أنه لا يمكن قياس كفاءة الأداة سوى في سياق استخدامها، بمعنى آخر، لا يوجد ثمة كفاءة بالمعنى المجرد. وتوخياً للبساطة، يمكن تمثيل هذا السياق على ثلاثة محاور:

- محور الأداء الفردي الذي يقيس الكفاءة التقنية لثنائي الإنسان-الأداة في سياق العمل الذي ينبغي معالجته.
- محور الأداء الجماعي الذي يسمح بالحكم على الأداء من حيث مدى تعزيزها للتواصل. إذ أن من شأن أداة معقدة أن تجلب أنماط استخدام ربما تضر بشكل فادح بالتواصل، فكل ميل للعمل ضمن مجموعة فرعية خاصة به، الأمر الذي يتنافى مع مبدأ الأداة ذاته. جدير بالذكر أن منحني الخبرة الجماعية أبطأ بكثير عمومًا من منحني الفرد الوحيد.
- محور الأداء ضمن المؤسسة الذي يسمح بتحديد دور الأداة و/أو الأسلوب في تحقيق أهداف الجودة العامة التي تسعى إليها المؤسسة، والتي قد تسفر عن تبني استراتيجيات استجواز من شأنها تأمين المكانة المثلى للمؤسسة على الصعيد العالمي، على حساب هذا المشروع أو ذاك على الصعيد المحلي. ويبدو منحني اكتساب التجربة الخاص بالمؤسسة إزاء استراتيجية الأداة أكثر بطأً، بيد أن العائد على الاستثمار أهم بكثير.

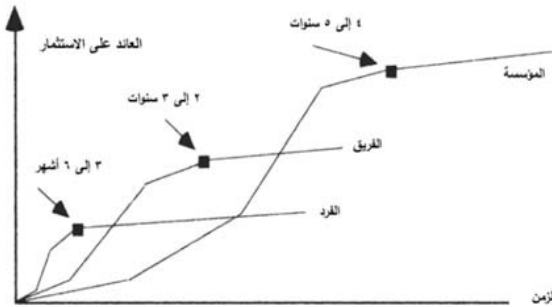
ويعد التقدم والنمو على هذه المحاور جوهر إشكالية نماذج نضج التطوير [٦].

هذا وقد يؤدي الاقتصار بالاهتمام بمحور واحد لا غير، وهو المحور

الفردى عادةً، إلى خيارات غير موفقة، بل كارثية أحياناً، وذلك حينما يكون محور آخر هو الذي يحقق الانتاجية. وهو أمر متكرر مع أدوات التصميم المساعدة التي هي أدوات تواصل في المقام الأول.

ثمة حقيقة لا بد من إدراكها، وإن صُعِبَ الاعتراف بها أحياناً، وهي أن جميع هذه الظواهر تتبع قوى متحركة بطيئة، لا يملك المرء سوى تأثيراً محدوداً عليها، إذ أن هناك مركز جاذبية كائن جماعي هو ما ينبغي تحريكه (أي كامل فريق العمل) [٢٠، ٢١].

ينبغي أن تساق السياسات المتعلقة بهذا الشأن بوعي عميق، وفطنة كبيرة وفي بعض الأحيان، بالكثير من الحزم.



٢- اختيار الأدوات:

أ- طبيعة الأدوات- قد تكون الأداة أحادية النشاط (محرر نص، المترجم للغة الآلة، إلخ...) أو متعددة الأنشطة (قاموس البيانات، إدارة بنية البرنامج أو تكوينه، إلخ...) وذلك حسب ارتباطها بنشاط محدد بدقة أو مجموعة من أوجه النشاط المختلفة.

ب- طبيعة المشاريع- يتم تحديد طبيعة المشروع من خلال ثلاث مجموعات من المتغيرات أو العوامل: حجم المشروع، ونمط توزيع النظام البرمجي، ونوع البرامج السائد.

ويعد حجم المشروع عاملاً رئيسياً في اختيار الأدوات. فلن يكون لمشروع صغير الحجم سوى علاقة محدودة بالبيئة المحيطة به، في حين أنه سيكون على برنامج ضخم إدارة سيل عظيم من المعلومات بين مختلف المكونات الفاعلة في البيئة التنظيمية.

ويمكننا تمييز ثلاث مستويات:

- المشروع المحدود الذي يتولاه فريق واحد مكون من ٥ إلى ٧ أشخاص على مدى عامين إلى ثلاثة أعوام، أي ما يعادل جهداً قدره ٢٠ فع لإنتاج يعادل حوالي ١٠٠ كسم. وتركز الأدوات في هذه الحالة على مواطن حاجة فريق العمل المحددة.

- المشروع الكبير الذي يتطلب ٣-٤ فرق عمل على مدار عامين إلى ثلاثة أعوام، أي ما يعادل جهداً لا يزيد عن ١٠٠ فع لإنتاج مالا يزيد عن ٥٠٠ كسم. وينبغي لمشروع من هذا القبيل والذي يمثل استثماراً هاماً، أن يلتزم بدقة بمعايير المؤسسة وتفاصيل التنفيذ؛ ويعني صعوبة البنيان والتجميع، ومطلب سلامة التشغيل والأداء.

- المشروع الضخم الذي يتولاه ما يزيد عن خمس فرق عمل، ويحتاج لمقاولين فرعيين، متطلباً مشتريات ضخمة، وملزماً بتلبية المعايير الدولية (منظمة التقييس الدولية أيزو-٩٠٠٠، الاقتناء وتعزيز دورة الحياة المستمر CALS، إلخ...). وفي هذه الحالة سيزيد الجهد عن ١٠٠ فع لإنتاج يناهز ٥٠٠ كسم. ويقتضي الاستثمار الكبير الخاص بهذا النوع من المشاريع عُمر أو مدة حياة طويلة جداً (أكثر من ١٠ أعوام)، الأمر الذي سيفرض تغيير عدة تقنيات؛ وتكون هنا بنية النظام المعمارية وتجميعه بالغة التعقيد، كما يترتب على النظام البرمجي تلبية العديد من القيود، مع صعوبة تحديد الحل الاقتصادي المناسب. وتكون أوجه التواصل مع البيئة لمثل هذا المشروع أساسية ومكثفة.

١	أهداف البرنامج الحاسوبي	معالجة النصوص، جدول
٢	سرد المتطلبات	معالجة النصوص، أدوات تكوين نموذج الماكيت
٣	التصميم	أدوات التصميم، المحاكاة، أدوات النمذجة المخبرية أو الأولية
٤	البرمجة اختبار الوحدات	بيئات البرمجة، أدوات التحقق والاختبار
٥	التجميع/التكامل اختبارات التأهل	مكتبة الاختبارات، مراقب الاختبارات مراقب الأداء
٦	التثبيت	المكتبة ووحدة التكوين الخاصة بنظام التشغيل
٧	التشغيل وصيانة الإصلاح	أدوات القياس، وتثبيت الإصلاح والنسخ
٨		أدوات مستعرضة أو افضية على كل مراحل - قوائم البيانات - إدارة التكوين - إدارة المشروع أدوات ميتا (أدوات عليا) - دعم المنهجية - مكتبة إعادة الاستخدام

هذا ويملي نمط تقسيم أو توزيع البرامج خيارات أدوات محددة. إذ تتطلب حزمة من البرمجيات خاصة بمشروع محدود دوماً إدارة لتكوين بنيتها، وذلك لمتابعة أوجه تطور المنصة، كما ينبغي شدة الحرص حينئذ على مراقبة الجودة.

ويؤوّل نوع البرنامج الحاسوبي المصنّع إلى ترجيح كونه يعود إلى أحد أنواع البرامج الثلاثة أي البرامج من نوع ثوم وب، وسيلعب هذا الترجيح دوراً حاسماً إزاء خيار بيئات البرمجة ومكونات مراقبة المعالجة وأدوات الاختبار. فإن اخترنا مشروعاً ذا واجهات مستخدم هامة، يكون علينا اختيار مولد واجهات بيانية ملائم للغة التطبيقات. وإن قمنا بتصنيع برنامج حاسوبي لحظي أي يعمل بالزمن الحقيقي على طائرة، ينبغي ضمان أن المكون المنفذ اللحظي يتمتع بتصديق شهادة الطيران، إلخ...

تفرض إذن طبيعة المشروع نوعين من القيود:

- ضرورة إمتلاك أدوات محددة للمشروع؛
 - استثناء أدوات محددة (تجنب إفراط في الحجم).
- وحسب الضرورة والأولوية، يمكن وضع الجدول التالي.

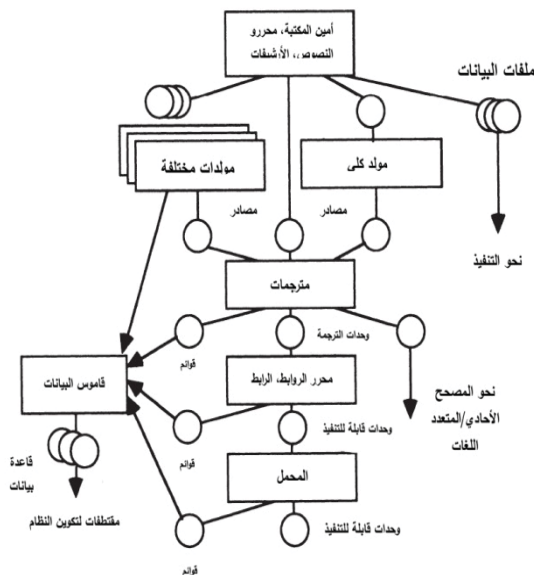
المراحل	المشروع المحدود	المشروع الكبير	المشروع الضخم
الأولى والثانية	• معالجة النصوص		أدوات الماكيت
الثالثة	• معالج الرسوم البيانية • الجدول الالكتروني	• أدوات التصميم • المحاكاة	أدوات النمذجة
الرابعة	بيئة البرمجة	أوجه الأداء	عدة بيئات
الخامسة	تدابير التغطية	أدوات الاختبار ومقاومة الانحدار	
السادسة والسابعة	أدوات من نوع (SCCS) (نظام التحكم بشفرة المصدر) وميك (MAKE)	أداة من نوع أوسي إس (نظام التحكم بالمراجعة)	• تجميع واختبارات النظام • إدارة النسخ • القياس
الثامنة	• قاموس البيانات • مقياس الجودة • إدارة المشروع الفردي	• إدارة تكوين البنية • إدارة التوثيق	إدارة مشروع النظام بما فيه من مقاولين فرعيين (مشاريع متعددة)

٣. ورش هندسة البرمجيات (بمساعدة الحاسوب)* :

أ) ورش التطوير - نظراً لمواطن التكرار والتداخل بين مراحل التطوير

* بالفرنسية (Ateliers de Genie Logiciel (AGL) بالإنجليزية (Computer Aided Software Engineering) : (CASE

وأنشطته، فإن هناك دوماً استخدام آني ومتزامن لعدة أدوات. فبيئة البرمجة تنطوي في الوقت الراهن على مايزيد عن عشرة أدوات مختلفة.



ومن الأهمية بمكان أن تكون جميع هذه الأدوات متماسكة ومتناسقة فيما بينها قدر الإمكان، وأن تتجنب تكرار المعلومات الذي يعد دائماً مصدر أخطاء جسيمة. وهذا التماسك والتناسق يجب أن يشمل على الأخص واجهات المستخدم أي التواصل بين الإنسان والآلة والأوامر الخاصة بالأداة (ويعني ذلك إظهار أو عرض هذه الواجهات باستعمال نفس الأسلوب ونمط التشخيص، والدعم على الانترنت)، ويشمل أيضاً، أشكال البيانات.

(ب) ورش وبيئات التطوير المتكاملة: نتحدث عن ورش أو بيئات التطوير المتكاملة حينما توجد بنية محددة مسبقاً تتيح تبادل المعلومات بين أدوات التطوير. وقد تكون:

١- قاعدة بيانات قادرة على التحكم بالكيانات المنبثقة عن أوجه نشاط عملية التطوير المختلفة. ونظراً لطيف الأدوات المتنوع الضروري لعملية التطوير، لا بد وأن تكون قاعدة البيانات هذه «مفتوحة» بحيث يستطيع المستخدم إضافة أنواع من الكيانات الجديدة ودمج أدوات جديدة. ينبغي إذن أن تكون الورشة أو بيئة التطوير مزودة بنموذج مجرد (نموذج مترفع)* موثق يستطيع مدراء بيئة التطوير النفاذ إليه لتمكين عملية تحديد المعايير والثوابت. ويتم التعبير عادة عن النموذج والنموذج المجرد له على شكل «علاقات الكيانات» (وحتى على شكل مخططات الصفوف المستعملة في البرمجة غرضية التوجه) المناسبة لهذا النوع من التمثيل.

٢- مجموعة من الواجهات البينية التي تتيح استقبال وإرسال المعلومات على شكل تبادلات معيارية تقليدية (مثل لغة XML وهي لغة الرقم القابلة للامتداد ذات القدرة على دعم هذا النوع من الواجهات)، بحيث تبقى لكل أداة حرية تبني نموذج بيانات مطوع لخدمة متطلباتها (ملاحظة: بنية البيانات الخاصة بالترجم إلى لغة الآلة مختلفة تماماً عن بنية محرر النص، وإن اشتركا في عناصر مثل الأشجار النحوية).

وقد ساد هذا النهج الأخير بلا منازع، بعد المحاولات الموءودة لمشاريع حكومية ضخمة انطلقت مع حمى حقبة الثمانينات PCTE في أوروبا، وSTARS وCAIS في الولايات المتحدة، وPCIS المشترك لدى كل من أوروبا الوسطى والشرقية والولايات المتحدة)، والتي انتهت جميعها بإخفاق مالي تام واسع النطاق.

* أي Meta Modelle وهو الذي يحتوي على إمكانيات للتعديل ولتغيير العوامل والثرابت والمعايير في الأداة دون أن يؤثر ذلك على عملها :

هناك أدوات وافرة معروضة في الوقت الراهن، وهي تتعامل على نحو جيد، مع لغة واحدة أو عدة لغات (C++, Java, C#, Visual Basic...) ومزودة بواجهات سهلة الاستخدام إلى حد كبير، كما أنها تمكن المبرمجين من التحرر من المهام اللوجستية التي لا غنى عنها، وإن كانت مهمة أحياناً لأنها تنطوي على الكثير من التكرار الذي يجعلها عرضة للعديد من الأخطاء.

باتت الهندسة اليوم حول شبكة الانترنت تتطور في سوق تنافس حرة، تكاد تسيطر عليها تماماً الولايات المتحدة الأمريكية، وهي تتمركز بشكل خاص حول لغات جافا وسي C#.

أخيراً، تجدر الإشارة إلى جهود كل من مؤسسة البرمجيات الحرة (Free Software Foundation) ومشروع غنو (GNU) لدي معهد ماساتشوستس للتكنولوجيا MIT، التي تزود منذ عدة سنوات مستخدمي نظام التشغيل يونكس بأدوات أساسية جيدة.

الفصل الخامس

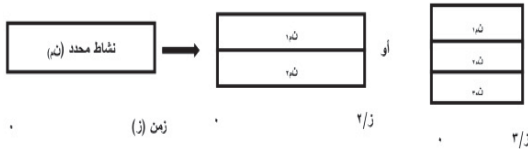
حركية دورة التطوير وديناميكيته وتنظيمها

١. حالات الاتزان والتقلب

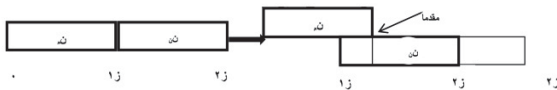
تعطي دورة التطوير الإسمية المستعرضة في الفصل الرابع رؤية راکدة للتطوير، بيد أن الأمور ليست بهذه البساطة على الصعيد التطبيقي، فكل شيء يتحرك.

هذا وهناك بعض المراحل التي يمكن إجراؤها بالتوازي، مثل المرحلتين الأولى والثانية، في حين أنه ينبغي لمراحل أخرى أن تكون منفصلة تماماً، مثل المرحلتين الرابعة والخامسة. ونلاحظ في الواقع في جميع الأحوال تقريباً فترات تداخل متفاوتة الطول بين المراحل نظراً لكون الانتقال بين هذه المراحل تدريجي بشكل عام.

ويحرص المقال الرئيس بصفة عامة على انقضاء فترات العمل - الذي يسفر عن تكاليف مستمرة - في أسرع وقت ممكن. من هنا تنبثق ثلاث تحديات:



التحدي الأول: إلى أي مدى نستطيع تجزئة مرحلة ما؟



التحدي الثاني: متى نستطيع المباشرة بمرحلة ما في حين لم تنته بعد المرحلة السابقة ها؟

من جهة أخرى، إن كان تاريخ بدء مرحلة ما محدداً بشكل واضح (هناك شخص واحد على الأقل مسؤول عن ذلك)، ربما لا يسهل الإعلان عن انتهاء مرحلة ما إن لم نحقق المعايير الكمية التي تحدد نهاية هذه المرحلة، ومن هنا ينبثق:

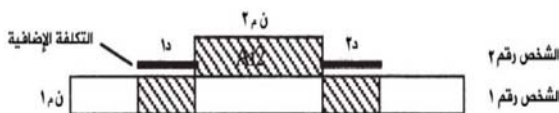
التحدي الثالث: متى ينبغي التوقف؟

معايير التجزئة:

لتجزئة نشاط محدد ما (م) ينبغي:

- أن نستطيع تمييز أجزاء مستقلة ضمن هذا النشاط، تسمح لنا بتكليف شخص إضافي بها.
- أن يكون لدينا حقاً شخص يتمتع بالمؤهلات المطلوبة لتولي هذا النشاط الفرعي.

وفيما يلي الرسم التوضيحي لمبدأ تجزئة النشاط:



- تمثل ١د التكلفة الإضافية المترتبة على الشخص الأول لكي يستطيع الشخص الثاني مباشرة العمل.

- تمثل ٢د تكلفة التكامل لكي يتمكن الشخص الأول من استرجاع تماماً كل ما قام به الشخص الثاني.

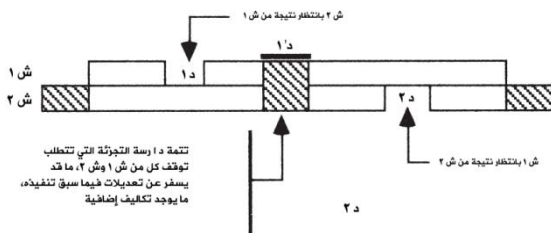
ولكي تكون عملية تجزئة النشاط مربحة، لا بد من تحقق ما يلي:

$$١د + ٢د > ن م \quad \text{حيث } ن م = ن م + ن م$$

وفي حال شاب تحليل عملية التجزئة خلل ما، أو جرى التسرع فيها،

تمثل أوجه نشاط ثغرات تطوي على تكاليف و/أو مواطن التأخير التي لا

يمكن تعويضها كما يوضح الرسم التالي:



يصبح حينئذ شرط تحقق الربحية، بعد عدد e من فترات الانقطاع:

$$N \geq \sum_{m=1}^{\infty} d_m + \sum_{m=1}^{\infty} d'_m + d_0 + d_1$$

لا بد إذن من توخي الحذر الشديد وحسن الاستعداد لكي تكون عملية التحزبة مريحة.

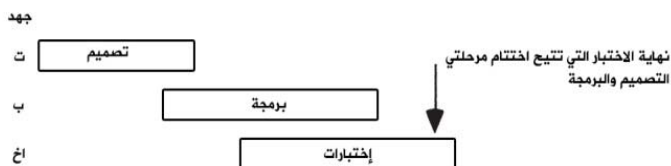
معايير التداخل والإنهاء:

التداخل ظاهرة حركية، تماماً كما التداخل الموجود في اشتعال محرك رباعي الأشواط. وتتوقف إمكانية حدوث تداخل على طريقة انتهاء المرحلة السابقة، وظروف انطلاق المرحلة الجديدة ومتطلباتها. هذه الدينامية تأتي على شكل منحني S تقليدي.

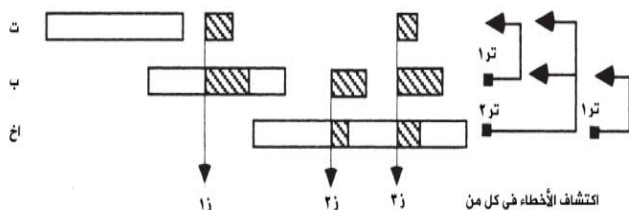
ويمكن اعتبار حالتين:

- المراحل أو أوجه النشاط المغلقة: تتوقف معايير إنهاء النشاط على النشاط وحده، فنستطيع تحرير الموارد تماماً بانتهاء المرحلة. وكمثال على ذلك مرحلتي توصيف المتطلبات والتجميع.
- المراحل أو أوجه النشاط المفتوحة: لا يمكن إعلانها منتهية إلا بانتهاء مراحل أو أوجه نشاط أخرى أيضاً. ينبغي هنا الاحتفاظ بفريق متبقٍ أو، على الأقل، الاحتفاظ بإمكانية تغيير فريق العمل كلياً أو جزئياً.

وكمثال على ذلك مرحلتي التصميم والبرمجة، فهي مراحل لا يمكن اعتبارها منتهية حتى إجراء آخر اختبار بنجاح. في عالم مثالي وخالٍ تماماً من الأخطاء، يكون الوضع كما يلي:



تكون إذن المحصلة المثلى للجهود: ت + ب + ا
إلا أنه في عالمنا البعيد عن الكمال، يكون الوضع أشبه بما يلي:



هناك نوعان من التغذية الراجعة:

- التغذية الراجعة أحادية المستوى ونرمز لها ترا: وهي اكتشاف خطأ في مرحلة، من مراحل التطوير يعود سببه إلى المرحلة السابقة مباشرة. وهو ما يحدث في الزمن ١ و ٢. وكمثال نموذجي لهذا النوع اكتشاف خطأ أو برنامج يعمل في مرحلة البرمجة واختبارات الوحدات (المرحلة الرابعة) يرجع سببه إلى المرحلة السابقة أي مرحلة التصميم (المرحلة الثالثة) يكشف عن عدم اتساق في

التصميم التفصيلي.

- التغذية الراجعة ذات مستويين ونر مزلها تر ٢: وهو ما يحدث في ز ٣
ويتطلب العودة إلى مرحلتين سابقتين مثل البرمجة والتصميم. وهي
الحالة النموذجية لخطأ يكتشف في عمل واجهة المستخدم والحاسوب
ويقتضي تعديل في مرحلة التصميم، ومن ثم جميع البرامج المتعلقة
بها في المرحلة البرمجة.

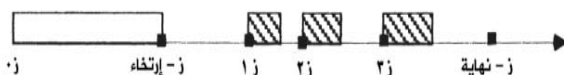
في حال الكشف عن عدد (ع) من الأخطاء، تصبح المحصلة:

$$\sum_{m=1}^E d_m^1 + \sum_{m=1}^E d_m^2 + \sum_{m=1}^E d_m^3 + \dots + x$$

ويصبح موعد النهاية من العمل في المرحلة المعتبرة:

$$\text{زنهاية} \leftarrow \text{زنهاية} + \sum_{m=1}^E \Delta_m \quad \text{حيث } \Delta_m \text{ التأخير الناجم عن الخطأ } m.$$

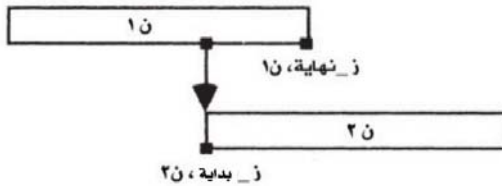
ويكون شكل المدة الزمنية أو العبء لمرحلة مفتوحة من مراحل تطوير
المنتج على النحو التالي:



ويُعرف موعد الإرخاء بأنه الموعد الذي كان مخططاً لتحرير الموارد
الخاصة بالمرحلة.

يتضح إذن إنه كلما ابتعد موعد الكشف عن خطأ عن لحظة زمنية
ز، كلما بات من الصعب، وقد يكون من المستحيل، تصحيح هذا
الخطأ، لمجرد كون الموارد قد خصصت لمواطن أخرى ولم تعد متاحة.
يفضل إذن الكشف عن أكبر قدر ممكن من الأخطاء قبل تاريخ تحرير
موارد المرحلة، والكشف عن تلك الأخطاء الأخرى قبل تفرق أعضاء فريق
المرحلة: وهو دور تداخل المراحل.

تأتي عملية التداخل على الشكل التالي:



حين يبدأ النشاط ن ٢، حكماً لا يكون ن ١ قد انتهى بعد. وثمة عدم يقين إذن حول تاريخ _____ نهاية ن ١ لاسيما وأن التداخل ضروري. وحتى لينطلق ن ٢ فعلياً، ينبغي تكوين نواة فريق، وتخصيص موارد بشكل مسبق للقيام بعملية التدرج في زيادة أعباء الفريق. وتورث حينئذ جميع تحديات التجزئة التي سبق الوقوف عليها، مع وجود عامل مفاقم، وهو كون المراحل منفصلة منطقياً، ما قد يملّي أوجه تنظيم متباينة. ويتضح أنه في حال لم يكن قد تم الإعداد جيداً للعملية من قبل أفراد متمرسين، يتعاظم خطر حدوث أوجه خلل مكلفة.

أما العامل المحدد لنجاح هذا النوع من العمليات فهو:

- من جهة، ثبات التوقعات مثل أزمّة نهاية المراحل وتكاليف المراحل، الأمر الذي يقتضي إدارة مشاريع بالغة الدقة؛
- من جهة أخرى متابعة منتظمة لمعدل تغيير جميع العناصر التي يتم تسليمها، ما يتطلب كذلك إدارة بالغة الدقة لتكوين بيئة البرنامج الحاسوبي.

من الضروري إذن توافر آلية قياس لتتبع تطور مختلف العمليات المتفاعلة مع بعضها بعضاً. وينبغي أن تكون هذه الآلية ذات دلالة ودقة إزاء الظواهر المرصودة، لا سيما وأنه لا غنى عن هاتين السمتين لضمان ثبات التوقعات. ولغياب المقاييس الحقيقية (الذي وجودها غاية بعيد من غايات هندسة البرمجيات) من الأسهل في بعض الأحيان التعويض عنها باللجوء

إلى تحديد مؤشرات للميل أو التوجه. وفي ضوء جميع هذه المعلومات التي ينبغي أن يبقى عددها معقولاً، يقرر رئيس المشروع ما إذا كان سيعتمد التوقعات المرسومة.

وفي مشروع نظام برمجي مدار بشكل جيد، تسير الأعمال من مرحلة لأخرى على نحو مستقر، بيد أن حالة الاستقرار هشة بشكل عام، إذ أن اتخاذ بعض القرارات غير الموفقة، أو الإفراط في التفاؤل، أو الافتقار إلى صفاء التفكير، هي أمور كافية لكي يتحول انسياب العمليات إلى حالة اضطراب فورية، تسفر لا محالة عن تراجع معدل الإنتاج. ولا يكون استقرار سير العمل ممكناً إلا حينما يكون ببيان النظام البرمجي سليماً، وأسلوب تنظيم المشروع ملائماً لتحقيق غرضه.

٢ - توصيف المتطلبات

من منظور عملية تطوير دينامية، نرى أن السمة الأولى لعملية توصيف المتطلبات هي أن تتطوي على توقعات متوسطة - بعيدة المدى.

وفي سياق حزمة البرمجيات، فإن الأمر يتعلق بتوقع سليم لحالة السوق، والاستعداد لتوزيع نسخ متتالية في فترات زمنية منتظمة إلى حد ما تعمل مع طيف متنوع من التجهيزات.

في حال البرمجيات الجاهزة (مفتاح باليد)، يمكن أن يكون مدى التوقعات من ثلاثة إلى خمسة أعوام، وهو نطاق واسع في سوق تشهد تطورات سريعة مثل سوق البرمجيات، في حين تكون مدة حياة النظام طويلة بشكل عام، أي تعادل ١٥ عاماً فأكثر.

فلنتقف عند تأثير توصيف المتطلبات على كل من حركية ودينامية وتنظيم دورة حياة النظام على التوالي:

أ- سياسة الاقتناء (المشتريات): وهي السياسة الخاصة بالصنع أو الشراء. فأسرع وسيلة لتطوير وظيفة ما هي شرائها جاهزة حينما تكون متوافرة، ومن ثم ضمها إلى النظام الحاسوبي الذي نقوم بتطويره. وفي جميع الحالات، يُنصح بتحديد وجود مصدر ثاني للوظيفة، و، بمصطلح

تقني، تغليف هذه الوظيفة في حال كان يوماً ثمة ضرورة لتغيير المورد. وينبغي توخي الحذر من التقنيات التي تكون في آخر دورة حياتها لأن مطورها لن يتابع الاستثمار فيها، وكذلك توخي الحذر من التقنيات البازغة، حيث أن احتمال عدم نجاحها عال.

ب- إدارة المتطلبات: بالنظر إلى وجود عدم اليقين، لا بد من القدرة على تأجيل أو إلغاء حاجة ما فيها إشكال في العائد على الاستثمار الخاص بها، والقدرة طبعاً على إضافة حاجة جديدة. لعل العامل الأكبر المتسبب في تأخير انجاز المنتج هو تطوير وظائف غير مجدية، وهو ما أطلق عليه العالم بي. بوهيم «الطلاء بالذهب» وهو أمر شديد الانتشار في المشاريع الكبيرة.

ولا يكون هذا التأجيل أو الإلغاء صعباً طالما لم تبدأ بعد عملية التطوير فعلياً، بيد أن الأمور تزداد تعقيداً في الحالة المعاكسة، إذ ينبغي وقف التصميم والبرمجة والاختبارات المتعلقة بالحاجة بشكل سليم. وهذا أمر حرجي في حال المتطلبات المسهبة، مثل سلامة التشغيل والأمن وأوجه الأداء.

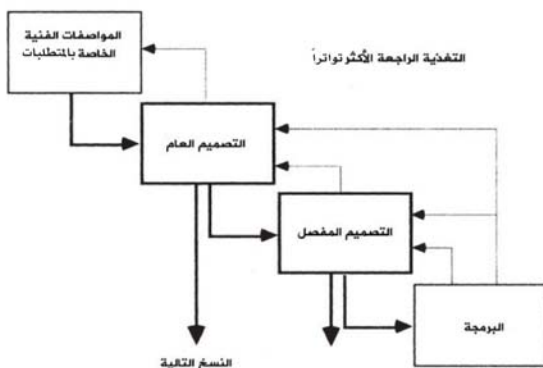
ج- ضبط التكاليف: يحدد توصيف المتطلبات القيمة الاقتصادية لمختلف أهداف التطوير، بيد أن تكاليف هذه الأهداف المتباينة تكون بالغة الغموض عندما نبدأ بتوصيف المتطلبات. وعند نهاية مرحلة التصميم، يكون هذا اللبس المحيط بالتكاليف قد تبدد جزئياً، ولا بد حينئذ من تحديث تحليل قيمة النظام البرمجي، بحيث يتم تحديد ما ينبغي حذفه أو تغييره أو حتى إضافته، وإعادة التفاوض في نهاية المطاف على الميزانية المخصصة في البداية.

جدير بالذكر أن تنظيم الجهود الخاصة بالتطوير هو نتيجة مباشرة لعملية تحليل المتطلبات. ومن المعتاد أن تكون جميع عناصر النظام

موصفة لتكون أقرب ما يكون إلى الكمال ومحسنة على نحو لا فائدة منه- وهي أيضاً ظاهرة الطلاء بالذهب- الأمر الذي يسعد المبرمجين الذين تكاد هذه النزعة تكون سمة نفسية يتميزون بها. وفي حال برامج من النوع برمج-ب ذات الواجهات المهمة مع البيئة المحيطة بهم، لا بد من التحكم بحزم بجميع هذه الواجهات التي يصعب جداً التحقق منها ومكاملتها مع النظام. ويعد استخدام الماكيت والمحاكاة، مصحوباً بسيناريوهات الاستخدام التي يصادق عليها المستخدمون الحقيقيون، وسيلة جيدة لتعريف الكل بالبعد الاقتصادي. ذلك أن اللبس المحيط غالباً بحدود النظام البرمجي، مصحوباً بالتفاؤل الطبيعي الذي يتسم به خبراء تقنية المعلومات، كثيراً ما يولد إشكالات - فهذا مدخل كبير لحالات عدم الاستقرار- وكوارث اقتصادية في نهاية المطاف.

٣. التصميم وبنية النظام الهيكلية

يعد بنية النظام البرمجي، نظراً لمكانته المحورية في السياق الاقتصادي الإجمالي لدورة حياة النظام، المنظم الرئيس له.



لا تتوقف ظروف تنفيذ النسخة الأولى فقط على صحة بنية النظام، بل كذلك النسخ التالية أيضاً وحياة النظام البرمجي الإجمالية في نهاية المطاف.

١ - طبيعة بنية النظام- ينجم بنية النظام عن ثلاث مجموعات

من القيود:

القيود الفنية المرتبطة بطبيعة النظام ومهامه.

القيود التقنية المرتبطة بوسائل وأدوات التصنيع المتاحة.

القيود الإنسانية المرتبطة بخبرة المستخدمين وبمدى نضج مؤسسات التطوير.

وتعد هذه القيود، والحلول الوسيطة الناجمة عنها، جوهر مراحل توصيف المتطلبات والتصميم.

أما التحدي الذي يواجه مهندس بنية النظام، فيوجد بغض النظر عن التقنيات المتوفرة أو التي ينبغي اختراعها لمعالجته. فمن بين المعضلات التي تواجه المهندس مايلي:

هل بوسعه أو عليه تصور نظام بتجرد تام عن الظروف التقنية أو الإنسانية المتاحة، وهذا ينطوي على خطر الانتهاء بنظام غير قابل للصنع؟ هل بمقدوره أو هل عليه تصور نظام كنتيجة للتقنيات المتوافرة، وهذا ينطوي على خطر التشتت بين هذه التقنيات؟

ينبغي ألا يغفل مهندس بنية النظام البرمجية أبداً عن كون إحدى مميزات النظام البرمجي هي مرونته وقابليته للتكيف. وهذا يحتم عليه حفظ هذه السمة، فالبرنامج الحاسوبي هو دائماً الذي يتكيف لدى ارتقاء النظام.

وعند إعداد مواصفات البنية، لا بد دائماً من الفصل بين:

- وصف النظام كما يظهر في واقعه المادي، بما في ذلك أوجه النشاط سواء اليدوية أو الآلية التي يقوم بها النظام. هذا العنصر الأول هو النموذج المفاهيمي.

- الوصف المنطقي للنظام الناجم عن الخيارات التقنية التي هي بمثابة حلول توفيقية ما بين المقتضيات الخاصة بمهمة البرنامج الحاسوبي، وتكاليف-مواعيد الاقتناء والتطوير ومستوى التقنيات، والموارد البشرية المتوافرة، وهذا ما يسفر عن النموذج المنطقي. فيما يلي المراحل الرئيسية من هذا التحليل:

الوصف	
صنع النموذج المفاهيمي	١
نقوم هنا بتطوير نموذج يعكس الوضع الفعلي والحقيقي الخاص بالإجراءات والبيانات والأحداث والعلاقات التي تربطها.	
المصادقة على النموذج المفاهيمي	٢
نتأكد بمساعدة سيناريوهات وبالتعاون عن كُتب مع المستخدمين من دقة وصحة النموذج المفاهيمي.	
تكوين التعبيرات التجريدية المنطقية	٣
انطلاقاً من النموذج المفاهيمي، تقوم بدراسة مبدئية رامية إلى الكشف عن أوجه التجريد الذي سيسمح بالفصل ما بين الجوانب التقنية والإجراءات والبيانات الأساسية الخاصة بالنظام والتي ينبغي أن تكون مستقلة عن الخيارات التقنية.	
تنظيم البيانات	٤
استناداً إلى الأشياء والكيانات المستقرة التي تحدد الأنشطة الرئيسية للنظام، نقوم بتطوير بيئة البيانات الأنسب.	

٥	تنظيم العمليات انطلاقاً من الأحداث التي يستجيب لها النظام، نقوم بتحديد مختلف العمليات اللازمة وأوجه التسلسل.
٦	توصيف البنية المعمارية المنطقية إعادة تشييد النموذج المفاهيمي مع إدراج جميع القيوم التقنية والإنسانية والفنية فيه.
٧	التحقق من صحة النموذج المنطقي واعتماده نقوم بالتأكد من كون نتائج المرحلة الثانية لازالت صحيحة، وأن الخيارات الفنية والتقنية لا تزال تلبي متطلبات المستخدمين أو وكلاء المقاولين الرئيسيين. عندئذ يتم التصديق على بنية النظام من قبل جميع الجهات ذات الصلة.

جدير بالذكر أن نهج مهندس بنية البرنامج هو نهج إبداعي بامتياز، وأن الأدوات والأساليب المصاحبة لهذا النهج هي أدوات نمذجة في المقام الأول.

ويمكن أن تكون النمذجة:

- نوعية، أي وصفية وغير رسمية، وينبغي أن لا ينفي هذا عنها صفة الدقة والصرامة.
- كمية، أي أنها تسمح بإجراء بعض الحسابات الخاصة بتقدير الحجم، أو التأكد من التلاحم المنطقي في نماذج البيانات أو العمليات (طوابير الانتظار، شبكات بيتري ...).

هذه الأدوات لا يمكن أن تحل قط محل الإبداع. ليس كل من يود أن يكون مهندساً معمارياً يتحقق له ذلك. فممارسة

الملاحظة والتحليل، والإحاطة بالموضوع محل الدراسة وما نوليه من اهتمام، والنظر للموضوع من منظور تحديات شبيهة، جميعها عوامل محددة لمستوى أداء المهندس المعماري. وبعد جمع جميع الحقائق أو المتطلبات التي تتصف بالاستقرار، سواء كان في حالتها (البيانات)، أو في علاقاتها السببية أو التسلسلية (العمليات)، يقوم مهندس باني البرنامج بتجسيد ملاحظاته، وربطها بعلاقات وظيفية تسلط الضوء على السلاسل السببية والمنطق الذي يبرر ظهور هذه الصفات المصاحبة أو المتتالية. ويعد استخدام السيناريوهات وسيلة هامة للكشف عن مواطن الثبات البنوي. وينبغي إثراء هذه الدراسة توليف وتجريد، ومن ثم ترسيخ، وتحديد ما ينتمي إلى داخل النظام وما ينتمي إلى خارجه، وما يبقى في حيز مسؤولية المستخدمين.

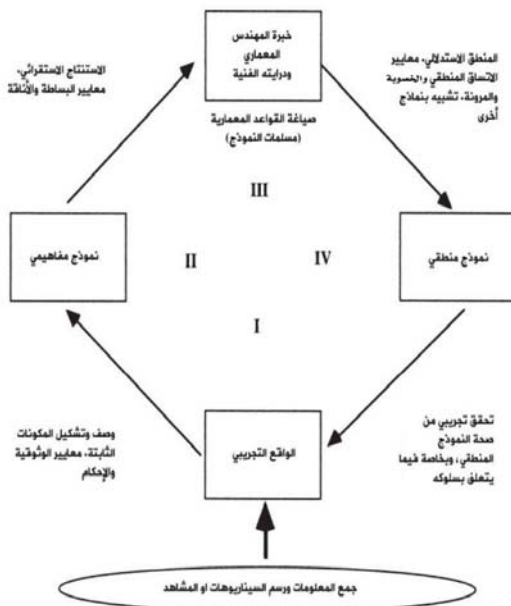
من الأهمية بمكان أن تتبع هذه العملية عن دراسة مفاهيمية باللغة الدقة تحدد بشكل محكم المفردات المستخدمة طيلة حياة النظام، بحيث تمكن مختلف اللاعبين من التواصل دون لبس. وكما هو شأن الرياضيات، ينبغي إيجاد رموز وقواعد بنوية للمفاهيم، مع الحرص على عدم التعامل مع الرمز على أنه مفهوم، لأن خطر التناقض يبقى قائماً. ومن شأن رموز غير دقيقة أو غير محددة بشكل جيد أن تسفر لا محالة عن عمليات معالجة مثيرة للريبة، كتلك التي حذر العالم بولزانو منها بشأن السلسلة اللامتناهية: المجموع $m = 1 - 1 + 1 - 1 + 1 - 1 + \dots$ قد تساوي صفر أو (1) أو $(\frac{1}{2})$ حسب طريقة وضع الأقواس (ملاحظة: لا حدود لمثل هذه العبارة!)، أو التمكن من إثبات عبارات عبثية على غرار:

$$1 = \sqrt{1} = \sqrt{(-1)^2} = (\sqrt{-1})^2 = (-1)^2$$

كما أن مجموعة رموز غير ملائمة للمعضلة التي ينبغي حلها، أو ببساطة غير مفهومة على النحو الصحيح من قبل مستخدميها (وبالتالي غير مستخدمة على النحو الصحيح) تمثل خطراً لا سيما وأنها تبدو

دقيقة. فالرموز أسلوب تركيب وترتيب جمل -تماماً كما الرياضيات بالنسبة للفيزياء- لا يضمن عدم قيامنا بكتابة عبارات غير ملائمة. يتطلب برنامج حاسوبي كبير تجزئة النظام إلى نظم فرعية، وهذا ما يتولاه فريق من مهندسي بنية النظام. فإذا كان (س) هو عدد المهندسين، فإن عدد جولات الحوار التي تُجرى فيما بينهم $E = S(S-1)$ ؛ وفي حال اتباعنا أسلوباً تنظيمياً ينطوي على وجود مهندس معماري وسيط، فإن عدد جولات الحوار هو $V = 2S$. وفي سياق جولات الحوار، فإنه من قيمة $S = 3$ فصاعداً، يكون الخيار الأرجح من الناحية الاقتصادية هو اختيار أسلوب تنظيمي ينطوي على مهندس وسيط يتولى مسؤولية كل الأمور التي تشترك فيها النظم الفرعية. جدير بالذكر أن ما ينجم عن ذلك من طبقة يتوقف على طبيعة الحل المتبع؛ إذ ينبغي أن يتحلّى الوسيط بنفوذ يقر به زملاؤه فيما يتعلق بالقضايا التي ينبغي معالجتها، كما ينبغي أن يكون ذا قدرة متميزة على التواصل. وفي سياق التداعيات، بوسعنا تصور المشاكل المعقدة التي تظهر حينما نختار أسلوباً ما لإدارة مشروع دون الاهتمام حقاً بالتحدي الذي ينبغي معالجته من خلال المشروع، فهناك ما يدعو إلى الاعتقاد بأن الإدارة في هذه الحالة ستكون غير مؤهلة لمعالجة المشاكل التي لا تعد ولا تحصى والتي سترفع إليها، وأنها ستوكل القرارات المهمة للجميع لمسؤولي المهام الدنيا. وهذا من أهم أسباب انهيار المشاريع الكبيرة.

٢- بناء النماذج: ينطوي النشاط المعماري في المقام الأول على تشييد النماذج كما سبق أن أوضحنا. ويمكن توضيح هذا النشاط من خلال الرسم التالي:



أن نقطة البداية في هذه الدورة إما أن تكون عند I حين نبدأ من نقطة الصفر لمنتج جديد كلياً، أو عند II حين نطلق من كيان ما موجود نود تعزيزه (الصيانة الارتقائية) أو إعادة صنعه (الهندسة العكسية). بالنسبة للمنظم الكبيرة، يتعذر عموماً إدراج كل ما نريد تضمينه في النموذج في آن واحد. وهذا يحملنا على خوض عدة دورات لإثراء النموذج البدئي بشكل تدريجي ومحكم. ومن المهارات التي لا تزال نادرة جداً والتي ينبغي توفرها في المهندس المعماري، مهارة القدرة على تمييز النظام الفرعي الذي سيتيح إطلاق العملية والقيام بدور المركز التنظيمي

الذي ستلتحم حولة بشكل تدريجي الوظائف التي تضمن الغطاء الوظيفي الكامل للنموذج.

من تحديات عملية النمذجة، تجنب كون قواعد البناء التي يتم اعتمادها بادئ الأمر، والتي تكون بمثابة مسلمات النظام، مفردة الارتباط بالواقع التي انبثقت منه، بحيث يعاد النظر فيها عند كل جولة تكرار. في المرحلة الأولى من النمذجة، تنطوي الصعوبة إذن على تجنب تبني أسلوب ”هيروغليفي“ نخلط فيه بين المفهوم، وما يمثله. ويعكس تاريخ العلوم والتقنيات وتشريح عدد من المشاريع التي تم إجهاضها، صعوبة هذا النوع من الممارسة.

ومن القيود التقنية التي لا فائدة من التجرد عنها، تلك القيود التي تشكل جوهر الحاسوب ذاته:

- مفهوم البيانات، الذي نربطه بشكل طبيعي بمكونات النظام الثابتة، وبالتمثيل الراكد لحالات النظام.
- مفهوم التعليمات والخوارزميات التي تحول البيانات، والتي سنربطها بشكل طبيعي بعمليات الانتقال بين الحالات، وإدارة الأحداث والتمثيل الحركي لمراحل ارتقاء النظام المتتالية.
- تشارك الأوامر والبيانات حيز الذاكرة في الحاسوب، ما يسمح لبرنامج ما بتصنيع برنامج آخر (وهي آلية ”الربط أو الإقلاع أو تمهيد التشغيل“)^x؛ ويتم التواصل مع العالم الخارجي بواسطة قنوات تقنية.

ويظهر لدينا من حيث النمذجة:

- نماذج البيانات (المفاهيمي والمنطقي) التي تقوم بوصف الحالات الثابتة.

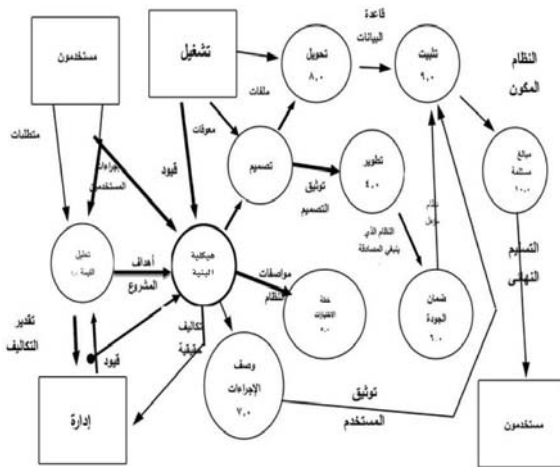
* Boot Strap :

- نماذج المعالجة التي تقوم بوصف تسلسل وتعاقب العمليات (السيرورات).

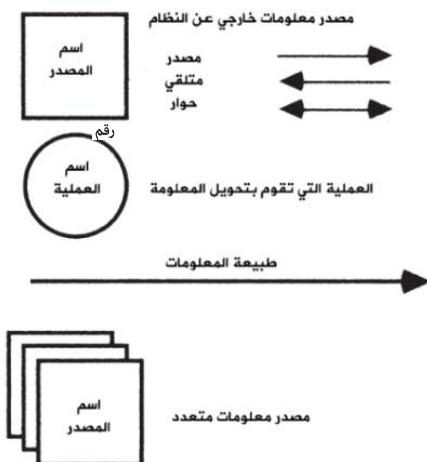
يضاف إلى ذلك، أن هذه الرؤية هي رؤية هرمية، وذلك حرصاً على القدرة على فهم كل من مستويات التجريد، التي ينبغي أن تبقى مستقلة.

٣- تمثيل النماذج: تتفاوت نهج التمثيل - وهي بيانية في المقام الأول - التي نجدها في أساليب التصميم [١٢، ١٥، ٢٠]، بيد أنها تستند إلى بعض المبادئ العامة التي سنذكر بها بإيجاز.

أ- مخططات السياق- وهي التي تصف علاقة النظام ببيئته. في هذا النموذج الشكلي، يمكن تمثيل عملية التطوير كما في الشكل التالي:



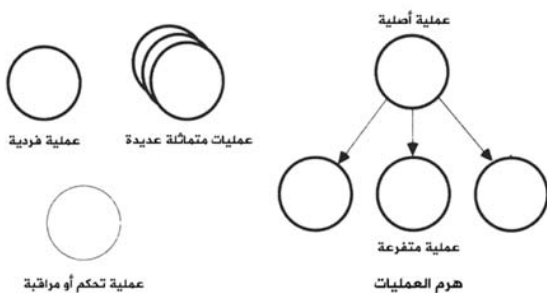
وفيما يلي الرموز الأساسية الخاصة بتمثيل مخطط السياق:



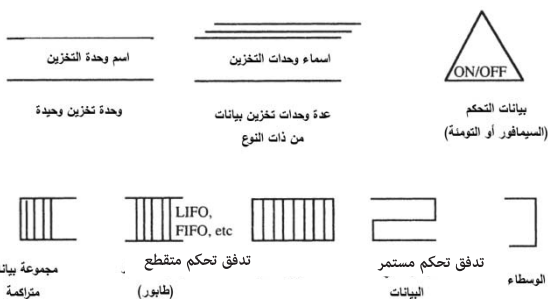
الجدير بالذكر أن الغاية من هذا المخطط هو رسم حدود النظام، وتجسيد العمليات الأساسية.

ب - مخططات تدفق البيانات: يمكن تعزيز مخططات السياق من خلال وصف بنية البرنامج الحاسوبي بدقة عند مستوى التجريد المطلوب باستخدام التمثيل بالرموز التالية:

- أنواع العمليات المختلفة:

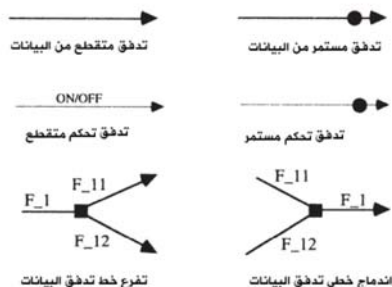


- البيانات الثابتة:



جدير بالذكر أن مجموعة دقيقة من العمليات تصاحب كلاً من هذه الهياكل.

- اتجاه انتقال المعلومات ونوعها



ويصاحب عملية إعداد مخططات تدفق البيانات توليد مجموعة مصطلحات للتعبير عن جميع مكونات هذه المخططات، وهو ما يسمى بقاموس البيانات.

ج- مخططات علاقات الكيانات: يُطلق على كل كيان أو وحدة فهرسة في مختلف مناطق تخزين البيانات اسم كيان خاص به، وهو الاسم الذي يُعرف به هذا الكيان في قاموس البيانات. وينطوي دور نماذج البيانات على:

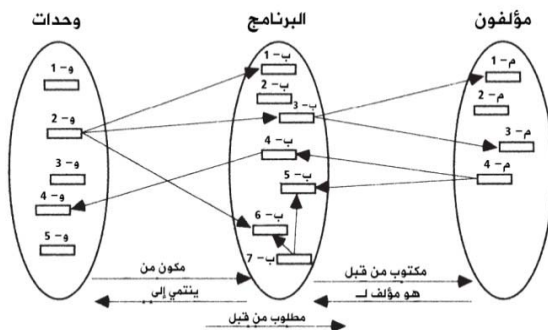
- وصف بناء أو هيكل الكيانات (الهيكل المفاهيمي والهيكل المنطقي والهيكل المادي في الذاكرة).

- وصف علاقات الترابط ذات الأهمية (أي تلك التي من شأنها تبسيط عمليات المعالجة أو تحسين عامل السلامة).

أما المستوى الأول من نمذجة البيانات فتتموضع علاقات الكيانات، وهو نموذج مفاهيمي خالص، منفصل عن أية تقنية وأي تمثيل للذاكرة. ويستند هذا النموذج رياضياً إلى نظرية الصفوف أو الفئات والعلاقات، حيث تُجمع الكيانات في مجموعات من الكيانات المتجانسة التي تكون صفوف الكيانات. نستطيع حينئذ دراسة علاقات الترابط الممكنة بين

مختلف الكيانات.

وبالنظر إلى الكيانات التي توجد في مشروع حزمة برمجيات، نجد العلاقات التالية:



ويمكن أن تكون العلاقات على درجات:

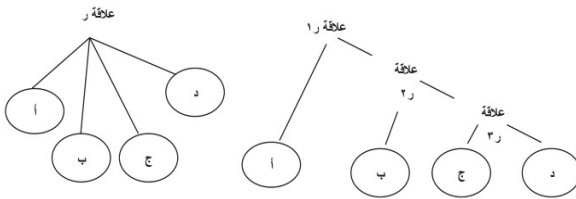
- أحادية: وهي علاقة جامعة تُعرف مجموعة ما (على سبيل المثال: العلاقة س هي - وحدة).

- ثنائية: بين كيانهين ينتمي كل منهما إلى مجموعة مستقلة، أو بين كيانات المجموعة الواحدة، كما هو مبين أعلاه.

- ثلاثية: بين ثلاثة كيانات، كما العلاقة بين برنامج مصدر والبرنامج الاثنائي المطابق له، وخيارات الترجمة الى لغة الآلة.

وبصفة عامة، تحدد دالة (أو تابع) ذات عدد ع من العناصر علاقة من الدرجة ع بين مجموعات تعريف مدخلات هذه الدالة.

ويمكن دائماً تبرير علاقة متعددة ما إلى عدة علاقات ثنائية: (انظر الشكل التالي).



يبد أن الأمر تطلب إضافة علاقتين مصطنعتين. وعلى الصعيد العملي، يسود استخدام العلاقات الثنائية إذ يمكن تمثيلها بمنتهى البساطة.

وتتميز العلاقة الثنائية بمجموعتين و دالتين:



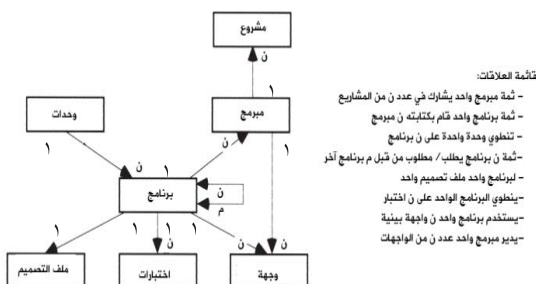
كما تتميز العلاقة بالطبيعة «الرئيسية»* للمجموعات التي تربطها، سواء:

- [١:١] أي دالة عادية واحد إلى واحد
 - [١:ن] دالة مباشرة ذات عدد (ن) من الصور (من واحد إلى كثير).
 - [م:١] ودالة عكسية ذات عدد (م) من الصور (من كثير إلى واحد).
 - [م:ن] ثمة (م) من الكيانات من مجموعة البداية لهم (ن) صورة.
- وتساعد سمة الطبيعة الرئيسية في اختيار نمط التمثيل المادي الأنسب.

* الطبيعة الرئيسية = Cardinalite :

بيد أن ثمة عناصر لا يمكن أن تكون لها صورة، فتكون الدالة حينئذ جزئية. والجدير بالذكر أن معظم الدوال المستخدمة في البرامج هي دوال جزئية. وباستخدام هذه المفاهيم المنطقية بالغة البساطة يمكن إضافة عدد كبير من المعلومات بشأن طبيعة الروابط القائمة بين الكيانات المختلفة. كما أن العمليات الكلاسيكية الخاصة بنظرية المجموعات $(U, \cap, \in, \subset, X)$ هي عمليات قابلة للتطبيق في هذا المقام.

من ثم يمكن تشييد رسوم بيانية معززة كما هو مبين:



في حال كان عدد الكيانات كبيراً وكانت هذه الكيانات بالغة الاختلاف فيما بينها، تتم إدارة الكيانات من قبل نظم إدارة قاعدة بيانات، بيد أن هناك العديد من التطبيقات، بما في ذلك تلك المتعلقة بالإدارة، التي يجد مهندس بنیان النظام نفسه مضطراً لوضع نموذج بيانات مصمم خصيصاً لملاءمة المسألة المطروحة. من هنا فإن الإحاطة الجيدة بنماذج البيانات الأكثر انتشاراً أمر مفيد [١٥]. فهندسة البيانات بعد بالغ الأهمية من أبعاد هندسة البرمجيات.

د - نماذج التتابع : تقوم نماذج التتابع بوصف :

الترتيب الذي ينبغي تنفيذ العمليات وفقه.

الأحداث التي يُحتمل أن تخل بتتابع العمليات.

سنقوم فيما يلي بعرض بعض المفاهيم الأساسية الخاصة بالتتابع بشكل موجز وفق العناوين التالية: الإجراءات أو السيرورات، التتابع التسلسلي، التتابع غير التسلسلي مع التزامن، والمداولات، والتشغيل الذاتي، والواجهات.

مفهوم الإجراء أو السيرورة - هو مجموعة من العمليات المعرفة بدقة، والتي تستوجب عدداً من «الموارد» حتى يتم تنفيذها. لكل إجراء اسم خاص به يميزه. وقد تتباين الموارد اللازمة لإجراء ما: من حيث البيانات والذاكرة وقدرة وحدة المعالجة المركزية، وقنوات الدخول-الخروج، ومدة الزمن الحقيقي، سيرورة أو إجراء آخر، إلخ...

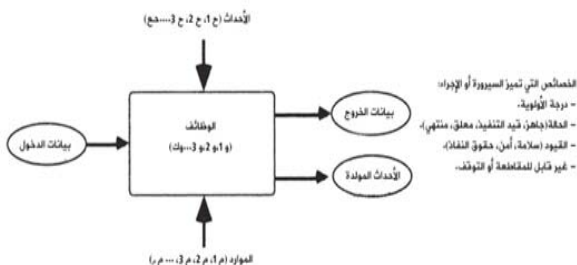
قد تكون الموارد مرتبطة بشكل ثابت بسيرورة أو بإجراء ما، كما قد يتم الحصول على هذه الموارد بشكل ديناميكي نظراً لاشتراك عدة عمليات أخرى فيها. وحين لا يتوافر أحد الموارد، يتم تعليق السيرورة، علماً بأنه لا يمكن استكمالها حتى يتاح المورد مجدداً.

ثمة نوعان مختلفان من التتابع ينبغي اعتبارهما أي:

- تتابع العمليات التسلسلي والذي يصف ترتيب تنفيذ الوظائف.

- تزامن عمليات النفاذ إلى مختلف الموارد.

نستطيع من هنا صياغة مخطط السياق كما يلي:



الخصائص التي تميز السيروورة أو الإجراء:

- درجة الأولوية.

- الحالة (جاهز، قيد التنفيذ، معلق، منتهي).

- القيود (سلامة، أمن، حقوق النفاذ).

- غير قابل للمقاطعة أو التوقف.

التتابع التسلسلي - يتم وصف تتابع الوظائف بمساعدة أساسيات

البرمجة البنيوية والتي تُكوّن البنية الأساسية للغة التصميم (والتي تعرف

بإيه دي إل ADL، لغة وصف المعمارية أو البنى). وفيما يلي الأوامر

الأساسية للغة من هذا النوع:

- الحلقات

- المعاملات Procedures (أو الروتينات الفرعية أو الوظائف)

- التحقق من السلامة

- التواصل عن طريق قنوات التواصل

ينبغي أن يكون لدى لغة وصف البرنامج (PDL) القدرة على التوسع

من خلال آليات توليد وحدات الماكرو، كما ينبغي اختيار قدرتها التعبيرية

وخصائصها البيانية وفق طبيعة البرنامج الحاسوبي المراد تنفيذه، ولغة

البرمجة الرئيسية المستعملة.

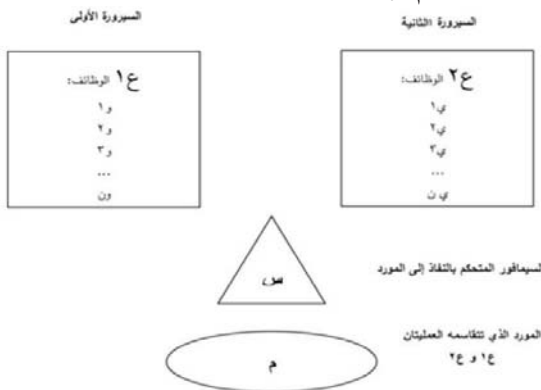
هذا وتتيح لغة وصف البرنامج، بمساعدة نموذج البيانات، وصف أنواع التجريد، والتي تنطوي على ربط محدد لبنى البيانات مع العمليات (مماثل لبنية المجموعة في الرياضيات). ويعد هذا التمثيل أساس طرق البرمجة "غرضية التوجه".

جدير بالذكر أن وصف النظام بواسطة لغة وصف البنى (ADL) يعد جانباً هاماً من جوانب التصميم المفصل.

التزامن - تتم مزامنة السيروورة أو الاجراء بمساعدة أسس التزامن البدائية التي تسمح بإدارة علاقات السيروورة ببيئتها ومحيطها. وتعد هذه الإدارة ذات دور محوري بصفة خاصة في هندسة النظم، حيث تسود البرامج التي من نوع برمجـ ب. وتتحقق المزامنة بواسطة كيانات باللغة البساطة تُعرف بالسيمافور أو التومئة (راجع إي. ديكسترا، في مقاله E. Dijkstra. Cooperating sequential processes).

فلنعرض مثالين لإظهار أهمية التومئة:

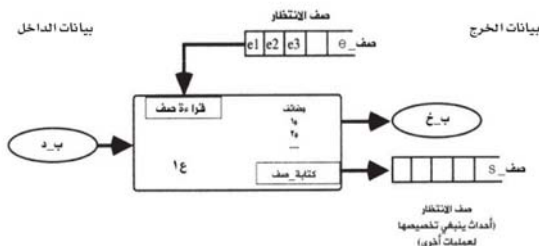
آ - حال تقاسم مورد ما:



لتنفيذ إلى المورد م، تطلب السيرورة ع ١ ذلك من سيمافور التنفيذ، ففي حال كان هذا الأخير "أخضر"، حازت السيرورة ع ١ بشكل مؤقت على المورد، فيصبح السيمافور "أحمرًا". حين لا يعود ل ع ١ حاجة من م، تقوم هذه السيرورة بتحرير المورد، وتحويل س إلى اللون "الأخضر"، فيصبح المورد مجدداً متاحاً لسيرورات أخرى. ويتم كل ذلك على أساس كون م في حيازة ع ١ أو ع ٢ بشكل متناوب. في حال أساءت السيرورة ع ١ إدارة السيمافور، بتركه في اللون الأحمر على سبيل المثال، يخفي المورد من النظام لأن النفاذ إليه لم يعد متاحاً، بما في ذلك مرات أخرى من استخدام ع ١.

في حال تم التنفيذ لعدة موارد م ١، م ٢، ... في آن واحد من قبل عدة سيرورات ع ١، ع ٢، ...، يتبع نظام إقفال السيمافور المقابل س ١، س ٢، ... اتفاقاً ما، وإلا وجدنا أنفسنا في مأزق تحوز فيه ع ١ على م ١، وع ٢ على م ٢، وتطالب ع ١ بالمورد م ٢ المأخوذ من قبل، بينما تطلب ع ٢ الحصول على م ١ ... حتى تعاق جميع السيرورات. وحدث مثل هذا الخطأ في برامج برمجة أمر ينطوي على مخاطرة تؤذن بكارثة. لذا فلا بد من الحيلولة دون حدوث مثل هذا الخطأ لدى تصنيع البرنامج.

ب- حال الأحداث: تقع الأحداث ذات الأهمية بالنسبة لسيرورة أو إجراء ما بشكل عام وفق صف أو طاوور انتظار مرتبط بالسيرورة نفسها. إذ أن بوسع السيرورة تنصيب أحداث عن طريق صف انتظار آخر. وفيما يلي رسم توضيحي لهذا المبدأ:



حتى تتمكن ١٤ من مراجعة صف الانتظار الخاص بها، ينبغي عليها تنفيذ وظيفة خاصة بشكل منتظم تسمح بمراجعة صف الانتظار. وتتوقف هذه المراجعة على طبيعة الأحداث التي تقع في صف e . في حال كان ينبغي معالجة الحدث قبل انقضاء فترة معينة، يكون من الضروري تفعيل الوظيفة قراءة صف قبل انقضاء هذه المدة. في حال ظهر حدث آخر من نفس النوع، ربما يتوجب معالجته مع الحدث الأول في آن واحد: حينئذ تقوم قراءة صف بمراجعة صف الانتظار بأسره لإعلام ١٤ بطبيعة كاملة من الأحداث (يجب دراسة التعقيد الخوارزمي الذي تنطوي عليه مراجعة صفوف الانتظار دائماً لأنه يؤثر على أداء البرنامج الحاسوبي بصفة عامة).

في حال وقعت الأحداث بسرعة مفرطة، تأتي ثمة لحظة يتشبع فيها صف الانتظار، فلا يعود قادراً على استقبال المزيد من الأحداث، الأمر الذي يضر بسلامة العمل (فقد معلومات). أخيراً، هناك بعض الأحداث التي قد تستوجب انقطاعاً فورياً لوظائف ون الجارية، بشكل يثير السيورة ١٤ بشكل لحظي (هذا حال الوقت الحقيقي).

ثمة برامج خاصة تقوم بإدارة الأحداث وصفوف الانتظار ذات الصلة

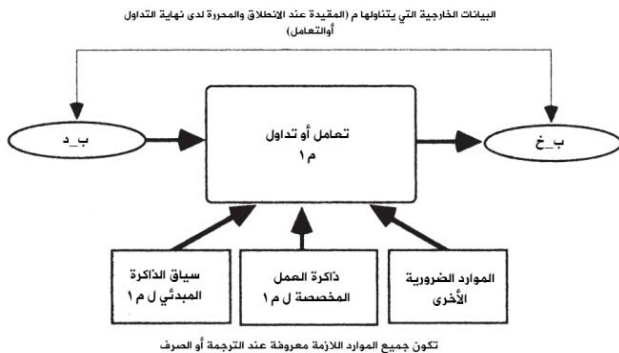
بها، تسمى بمراقب الطابور، وهي برامج بالغة الارتباط بالبنية المعمارية الخاصة بالمادة و/أو نظام التشغيل الكامن. وثمة طيف واسع من هذه البرامج الجاهزة للشراء، علماً بأنه ينبغي اختيارها وفق خصائص البيئة التي ينبغي للبرنامج الحاسوبي إدارتها (عدد وتنوع الأحداث، مدى التفاعلية المطلوبة، إلخ...). ذلك أن المراقبين هم برامج من نوع برمج تتطوي صياغتها على صعوبة خاصة.

تظهر هذه الأمثلة القليلة أن السيرورات، والموارد، والسيمافور المتصلة بهذه الموارد، وطوابير انتظار الأحداث، ومراقبي الأحداث، جميعها عناصر محورية في نمذجة بنیان النظام. ويتوقف كل من سلامة وثوقية تشغيل وأداء النظام البرمجي على الاستخدام الصحيح لهذه البرامج، التي ينبغي طبعا إدراجها في قاموس البيانات. وتجدر الإشارة إلى أن ترتيب معالجة هذه الكيانات عامل محوري لسلامة ودقة النظام. وينبغي أن تكون هذه الكيانات بالنسبة لبعض النظم بالغة الحساسية موضع نمذجة كمية بهدف حساب أبعاد البرنامج الحاسوبي (طول صفوف أو طوابير الانتظار، الموارد الاستباقية المخصصة بشكل ثابت للعملية، إلخ...) وضمان صفة الحتمية.

المداولات: هي نوع خاص من السيرورات الشائع استخدامها في نظم المعلومات التي تعالج قواعد بيانات هامة، وهي ما يُعرف بالسيرورات الخفيفة أو «الخيوط» أو سلسلة التعليمات في لغة يونيكس. تتطوي السيرورة المداولاتية على برنامج يتسم بالخصائص الأربعة التالية:

- اللاتجزئية أو الذرية: سلسلة الأعمال التي تنجزها المداولات لا تتجزأ، فإما أن يُجرى تعديل البيانات المعالجة من قبل العملية بالكامل، وإما ألا يُجرى أبداً. فلا توجد حلول وسطية، وهذه هي القاعدة العامة لمفهوم التعليمات غير القابلة للتجزئة.

- الاتساق: إن انتهى تعامل ما بشكل طبيعي، عادت قاعدة البيانات إلى حالة تناسق، وينبغي بصفة خاصة تلبية متطلبات شروط السلامة وعلاقات الارتباط الوظيفية.
- العزل: تحجب الأحداث الداخلية الخاصة بتعامل ما عن المداولات الأخرى التي يمكن تنفيذها بالتزامن، إذ لا تعتمد المداولات إلا على ذاتها.
- الاستدامة: أثر مداولة ما (م) هو أثر نهائي، بغض النظر عما يحدث بعد تنفيذ (م).
- وفيما يلي رسم توضيحي لمبدأ المداولات.



هذا ويتم تنفيذ المداولات بواسطة مراقب أو مراقب خاص يُعرف ببرنامج التحكم بالمداولات (مراقب المداولات) يقوم بتنظيم بيئة التعامل بحيث تتحقق الخصائص الأربعة (اللاتجزئية، الاتساق، العزل، والاستدامة) مهما حدث. وتتولى هذه البرامج بيئة الاتصالات برمتها (أي شبكة مواقع العمل)، فضلاً عن النفاذ إلى قواعد البيانات المختلفة بصفتها موارد. من هنا يتضح أن برمجة المداولات عملية بسيطة، الأمر

الذي يبرر الاهتمام البالغ بها. كما تقوم برامج التحكم بالمداولات بإدارة المداولات الموزعة، فهي تمثل آلية أساسية في نموذج العميل-الخادم (أو مخدم - زبون).

آلات الحالات محدودة العدد أو ذاتية التشغيل (أتومات): يقودنا مفهوم الإجراء أو السيروورة بشكل لا إرادي إلى التفكير في الحاسوب المثالي، الذي لا تعرف ذاكرته من أنواع البيانات سوى تلك التي عند الدخول والخروج، وتكون فيه التعليمات أو الأوامر مؤلفة من مجموعة الوظائف ف١، ف٢، ... فن. ويكون تتابع العمليات وفق آلية تسلسل تراعي محيط السيروورة (يتم النفاذ إلى الموارد والأحداث عبر «قنوات منطقية»)، وحالة نتيجة تنفيذ كل وظيفة من الوظائف في.

يمكن تنفيذ هذا السيناريو المثالي بشكل منهجي، بحيث يتم تجريد بعض «الآلات» فيه، المثيرة للاهتمام من الناحية الرياضية، التي ستقوم بتمثيل السيروورات وسلوكيات السيروورات شكلياً. وتعد هذه الآلات أو الأجهزة «المجردة» من أدوات نمذجة مصطلحات بيئة السيروورة، والخاصة بالتحكم بتطور نمو البرنامج الحاسوبي.

هذا ويتطلب تعريف الآلات المجردة المثيرة للاهتمام دراسة مفصلة لمختلف السيروورات، وإحاطة وافية بالسلوكيات المرغوبة للنظام، فضلاً بالطبع عن معرفة نظرية هذه الآلات.

ولهذا النهج فوائد عديدة أهمها:

- يمكن تفكيك الوظائف في إلى مجموعة من الوظائف الأكثر اختزالاً التي يمكن أن تقوم بدور التعليمات أو الأوامر، التي كثيراً ما تسمى بالأعمال. ويمكن أن يكون حجم هذه الأعمال من الصغر بما يكفي لفهمها واختبارها، والحرص على اتساقها بالخصائص الأربعة (اللاتجزيئية، الاتساق، العزل، الاستدامة). وسيتم استخدامها بشكل متزايد، مما يؤدي إلى درجة من الوثوقية تزداد بمرور الوقت.
- قنوات النفاذ المنطقية لمختلف الموارد هي كذلك بروتوكولات تواصل

بالبيئة، ويمكن تكييف مواصفات البروتوكول لطبيعة المورد. ويمكن تحديد عدد البروتوكولات والتحكم بشكل تام بتعقيدها.

- يتحدد تسلسل الأعمال وفق حالة عدد من «السجلات» التي تحتوي المعلومات اللازمة للتسلسل (وهذا هو، بشكل عام، مفهوم عداد البرنامج، وسجل المقاطعة، واسم الحالة... الذي يوجد في كل حاسوب).

- يمكن أن تتسم الذاكرة اللازمة لمختلف العمليات ببنية مخصصة (ad hoc)، يتم اختيارها وفق طبيعة المسألة، لتكون البنية الوحيدة التي تعرفها الآلة مثل: قوائم، مكدس، بنية شجرية، المخطط الخالي من الدورة، المصفوفة، إلخ... تختص الآلة إذن بهيكل محدد أو بنية محددة، ما يسهل عملية الاختبار والتشخيص حال حدوث عطل ما.

هذه الآلات المثالية هي الآلات ذات الحالات محدودة العدد (Finite State Machine). ولا مجال هنا لعرض نظرية هذه الآلات، بيد أن الدور الذي تضطلع به هو من الأهمية بما يجب عرض خطوطها العريضة بشكل موجز. وتتمتع هذه الآلات بسمتين بالغتي الأهمية في سياق هندسة البرمجيات هما:

- تستند هذه الآلات إلى نظرية رياضية دقيقة [١٣] تسمح بتعريف عمليات مجدية جداً، مثل تكافؤ وتشاكل آلتين (وهذا مفهوم جوهري لتمييز مواطن التجريد والوصول إلى التعميم)، وإنشاء الآلات المركبة (تتضافر عدة آلات بسيطة لتكوين آلة أكثر قدرة ذات مستوى مماثل أو أعلى من الوثوقية)، وتحديد الأولويات أو التدرج الهرمي (تضمنين مختلف طبقات الآلات التي نحدد لها ميزات خاصة، بعض الآلات قادرة على تشييد وإطلاق آلات ذات ميزات أقل).

- يمكن تمثيل هذه الآلات بمنتهى السهولة، إما بواسطة لغات البرمجة الكلاسيكية [١٤]، أو بواسطة الجداول التي يتم تفسيرها على نحو

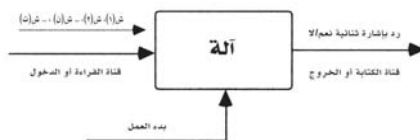
مخصص (ad hoc). فقد تكون "مترجمة إلى لغة الآلة"، وهي خاصية هامة جداً لمعالجة الخصائص المتشعبة التي تتسم بها النظم البرمجية. ويمكن أن يسفر تفعيلها عن إيجاد لغة شكلية (تعقب) هي بمثابة توقيع، الأمر الذي يسهل بشكل كبير معالجة المشكلات التي جرى تناولها في الفصل السابع الفقرة ٦. ويمكن التحكم بشكل كامل بالموارد اللازمة وفق تكرار طلبها.

تجدر الإشارة هنا إلى أن البرمجيات الأكثر وثوقية يتم تصنيعها جميعاً بواسطة آلات منطقية من هذا النوع، فسهولة تنفيذها يجعل منها عنصراً جوهرياً في التحكم بتعقيد البرنامج الحاسوبي. وفيما يلي مبدأ هذه الآلات:

إثر سلسلة لحظات ل، ل، ... ل_ن، بينها مهلة ما (ل_ن - ل_{ن-١})، تتلقى الآلة إشارات ش(١)، ش(٢)، ش(٣) ... ش(ن) .. وتصدر ردود ر(١)، ر(٢)، ... ر(ن).

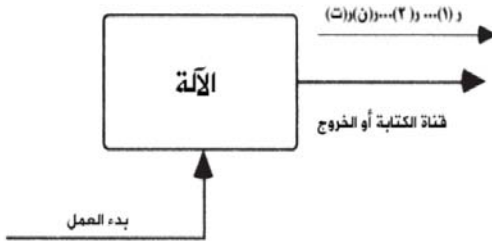
تباشر الآلة العمل عند اللحظة ل_٠. فلنعرض ثلاثة أمثلة:

- آلات الاستقبال أو التسلم: تكون الإشارات الواردة عبارة عن تركيب من الرموز المنبثقة عن مجموعة تمثل رموز أبجدية الدخول الخاصة بالآلة:

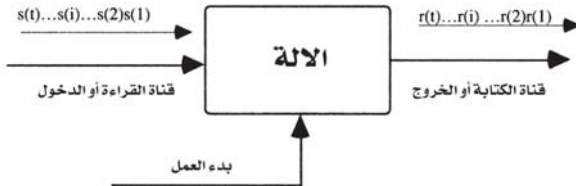


فإذا كانت سلسلة الإشارات صحيحة، كان الرد «نعم»، أما إن كانت السلسلة خاطئة، كان الرد «لا». فالآلة «حتمية» أي أن لكل سلسلة إشارات دخل دوماً الرد نفسه. وتشكل مجموعة الإشارات التي تستقبلها الآلة لغة دخول الآلة، ونجد هذا النوع من الآلات في محيط النظام: التحقق من معلومة ما، أو بروتوكول تواصل، أو ترميز المعلومات...
- آلات الإرسال أو الإصدار: آلة من هذا النوع لا تكون «حتمية»، وإلا

فتكون بلا أهمية. بيد أن هذه الاحتمية تخضع لقاعدة تقييدية:
فلسلة الرموز التي تصدرها الآلة تنتمي لمجموعة محددة تشكل
رموز الأبجدية الصادرة أو أبجدية الخروج من الآلة:



وتشكل سلسلة الإجابات الممكنة رموز الأبجدية الصادرة عن الآلة.
- آلات الأعمال أو المبدلات: هي الآلات الأكثر إثارة للاهتمام والأكثر
استخداماً، فهي تعمم الآلتين التي سبق عرضهما. فهي تتيح على سبيل
المثال تحقيق تطابق مزدوج أحادي الاتجاه بين «لغتين». وتستخدم
«ترجمات» مختلف طبقات بروتوكولات المنظمة الدولية للتقييس ISO من
واحدة إلى الأخرى أو إلى بروتوكولات أخرى، هذا النوع من الآلات. إذ
يتطلب كل نظام تعليمات فيه بعض التعقيد ولا تكون فيه رموز الأبجدية
ملائمة للمشغل البشري، هذا النوع من «التبديل»، وكذلك هو شأن أية
 $s(t) \in S$ و $r(t) \in R$ مع $t = 1, 2, \dots$



واجهة بين جهازين لا يحملان نفس اللغة يقوم أحدهما بتحريك الآخر.
هذه الآلات إذن بالغة الأهمية:

يُفترض أن يكون لدى الآلة عدد معين من «الحالات الداخلية» التي تنتمي لمجموعة محددة خاصة (Q). ويمكن أن تتغير حالة الآلة عند كل لحظة $t, t_1, \dots, t_n$... وفق الإشارات التي تُرسل إليها. ثمة دالة أو تابع انتقال الحالة (f) إذن كما يلي:

$$S(t+1) = f(q(t), t) \text{ مع } t \geq 0$$

تسمح بتحديد الحالة الجديدة لدى وصول الرمز $S(t+1)$. ويكون

نطاق تعريف الدالة (f) كما يلي: $f: Q \times S \rightarrow Q$

كذلك توجد دالة أو وظيفة خروج كما يلي:

$$r(t+1) = g(q(t), s(t+1))$$

حيث يكون نطاق التعريف: $g: Q \times S \rightarrow R$

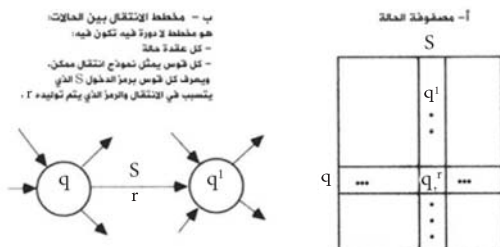
يتم إذن تعريف الآلة كما يلي:

$$M = (Q, s, R, f, g, q_{\text{initial}})$$

ويتكون الجزء الرئيس من الآلة من جدول الدوال f, g . ونجد على

الصعيد العملي نوعين من تمثيل هذه الوظائف:

(أ) مصفوفة الحالة (ب) مخطط الانتقال بين الحالات:



هو مخطط لا دورة فيه تكون فيه:

- كل عقدة حالة

- كل قوس يمثل نموذج انتقال ممكن.

ويُعرف كل قوس برمز الدخول S الذي يتسبب في الانتقال والرمز الذي يتم توليده r .

في التمثيل الموضح أعلاه، يرتبط الرمز r بحالتين (q, q') وهو ما نسميه بالانتقال. ويكون من الأسهل أحياناً ربط r بالحالة التي انتقلنا إليها للتو. هنا تكون دالة الخروج h أكثر بساطة ويكون نطاقها :

$$h: Q \rightarrow R$$

ينتشر استخدام هذا النوع من التمثيل بشكل كبير في ترجمة اللغات عالية المستوى إلى لغات الآلة. وفي جميع طرق التمثيل، تمثل السلسلة، $r(1)r(2)...r(n)$ (ي) $r...$ (ت) .. سلسلة أعمال ينبغي تنفيذها تلياً للتعليمات S التي تم إشعار الآلة بها.

وثمة ارتباط شديد بين درجة تعقيد النظام وعدد وتنوع الآلات اللازمة لوصفه. وحتى الآن أي عند هذه المرحلة، لم نضع أي افتراض بشأن التمثيل المادي لهذه الآلات التي يمكن ترجمتها إلى لغة الآلة أو تفسيرها، بل حتى ترميزها بشكل مايكرو*.

(هـ) الواجهات البينية (واجهات التوصيل أو الربط): ثمة

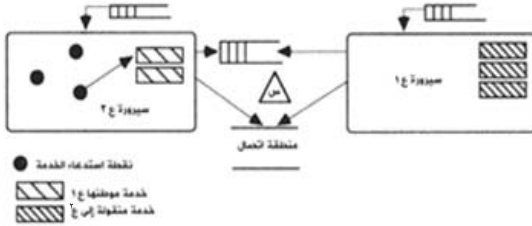
حاجة لواجهة بينية لدى الرجوع لإحدى الوظائف المتوافرة في البرنامج الحاسوبي من مواطن مختلفة في النظام، أو حينما تتطوي وظيفة ما على «مكون» ذي أهمية خاصة في البرنامج (ترشيح، غير مستقر...). وكثيراً ما يُشار إلى مثل هذه الوظيفة «بالخدمة» أو «الوظيفة البدائية».

فالواجهة إذن هي قاعدة تواصل بالغة الدقة تسمح باستدعاء هذه «الخدمة»: التي تقوم بدورها بالتحقق من أنه قد تم استدعاؤها وفق

القواعد.

* ترميز مايكرو = microcodees / تفسيرها = interpretees وترجمة إلى لغة الآلة = compilees :

وفيما يلي رسم شكل هذه الخدمات:



عندما تكون "الخدمة" محلية، يتم استدعاؤها بشكل عام بواسطة "مكدس أو كومة" وهذا يتطلب وضع قواعد تمرير المتغيرات الوسيطة (قيمة، مرجعاً... تكون موضحة الخصائص بشكل عام في أدلة استخدام لغات البرمجة). وحتى تتمكن عدة لغات من التواصل مع بعضها البعض، لا بد أن يقوم نظام التشغيل بإدارة هذا "المكدس" الذي لا ينبغي أن يقتصر على هذه اللغة أو تلك.

أما إذا كانت الخدمة متنقلة أو غير محلية، فهي تقوم عندئذ بتنفيذ أجهزة التزامن وآليات مثل Remote Procedure Call نداء الإجراء البعيد، كالتي نجدها في نظام شبكة الملفات في يونيكس NFS في نموذج المستخدم-الخادم. وربما يستغرق النفاذ إلى الخدمة قدراً هاماً من الوقت إذ ينطوي على عملية دخول-خروج.

لا بد من توجيه اهتمام خاص للواجهات البينية في مرحلة التصميم. وبالرغم من قربها من البرمجة، إلا أنه ينبغي تحديدها قبيل مرحلة البرمجة. إذ ينبغي أن يكون بالإمكان التحقق من صحتها بشكل مستقل.

ومن فصل عن وحدات النداء. كما ينبغي فهرستها على نحو دقيق في قاموس البيانات. فالواجهات هي بمثابة لبنة الأساس في بنية النظام. ويتطلب تسلسل بعض الواجهات وصفاً بمساعدة آلة ذاتية التشغيل FSM. وينبغي أن يتم أي تطوير للواجهة على نحو متساوق* أو متوافق مع بعضها بعضاً (وهذا تعميم بالمعنى الدقيق للكلمة) بشكل يتم الحفاظ على ما تم برمجته حتى هذه المرحلة.

ويتوقف النفاذ إلى خدمة أو أخرى على تنظيم النظام، وحقوق النفاذ الفردي الخاص بالوظيفة التي تقوم بالنداء، و الحقوق العامة على صعيد الخدمة. الجدير بالذكر أن إدارة هذا الحيز من الأسماء مكون أساسي في أي نظام برمجي ذي حجم معين، فهو يشكل الآلية المركزية الخاصة بعمليات أمن/سرية المعلومات التي يحفظها النظام. ومن نماذج اللغات التي أنشئت على أساس مفهوم الآلة ذاتية التشغيل FSM:

User interface terminal specification and description language [١٣،١] (لغة المواصفات والوصف الخاصة بواجهة المستخدم الطرفية SDL)، واللغة البيانية الخاصة بمخططات انتقالية الحالة، التي تم دمجها في لغة النمذجة الموحدة الخاصة بمجموعة إدارة الكيانات (UML of OMG).

٤- تعزيز أو أمثلة النماذج إلى أفضل حد ممكن. قد تُغيّر

صيغة النموذج الأولى طبيعة الاعتبارات التي قد نضعها للنموذج، إذ بوسعنا حينئذ اتخاذ أحد مسارين إضافيين:

- ينطوي أحدهما على تمثيل ووصف جميع كيانات وسلوكيات العالم الحقيقي كما تتضح بالملاحظة على أدق نحو ممكن.
- أما المسار الآخر فينطوي على العمل على التمثيل ذاته بهدف التوصل

* متساوق compatible :

- للصيغة الأفضل من الناحية الاقتصادية وفق الشروط العامة التي تحكم النظام البرمجي.
- ولتعزيز النموذج قدر الإمكان أو لأمثلته، ينبغي:
- التقليل من تنوع الكيانات.
 - البحث عن مواطن تناظر وتضافر البنى؛
 - عزل ما لا يشترك في شيء مع آخر ويتسبب لإشكاليات محددة؛
 - تمكين الصيغة وتعزيز قدرتها، بمعنى عدم السعي وراء أكثر صيغة مدمجة، مع تجنب إشباع بنى البيانات والتسلسل.
- بهذه الخطوط العريضة الأولية، يصبح بالإمكان تناول المعضلات الشائعة المتمثلة في:
- سلامة التشغيل
 - الأمن
 - أوجه الأداء.
- ولا يمكن استيعاب هذه الجوانب إلا بالاستعانة بتمثيل النموذج وبالعودة على الأرجح إلى الصياغة المبدئية للنموذج.
- في هذه المرحلة فقط يصبح بالإمكان البدء في الدراسة التحليلية المفصلة التي تسبق عملية البرمجة بشكل مباشر.
- أ- التقليل من تنوع الكيانات: لا يتم تمثيل الواقع إلا من خلال بعض الهياكل «الخاصة» التي ندرك قواعد معالجتها:
- القوائم، والملفات، والمكدس والهياكل الشجرية، والبيانات الحلقية،
 - البيانات اللاحلقية... فيما يتعلق بالجزء الثابت.
 - مجموعات شتى من الآلات المؤتمتة فيما يتعلق بالجزء الديناميكي.
- فمن شأن ذلك تسهيل عملية تمييز مواطن التماثل ومجموعات الوظائف، إلخ...
- يمكن القول إن الأمر بمثابة تطبيق للقاعدة المنطقية القديمة لموس

أوكام أو سكين ”أوكام“ ، والتي تنطوي على إزالة كل الكيانات الوسيطة التي لا يتضح ارتباطها بالحقيقة الواقعية بشكل واضح (أي عدم مضاعفة المجموعات).

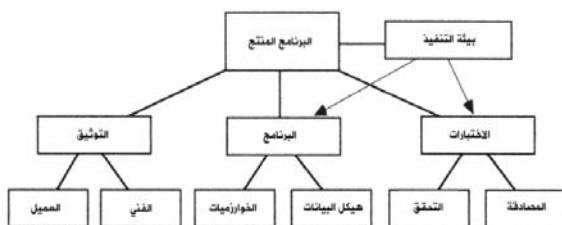
ب - مواطن التناظر والتشابه. وسط مختلف السيرورات أو الإجراءات، يمكن تمييز وظائف أو مجموعات من الوظائف المتماثلة. وتعد الصياغة على شكل آلات ذات الحالات المحدودة وسيلة فعالة للكشف عن مواطن التماثل. إذ يمكن من خلال التعميم فرض التشابه (وهو أسلوب كلاسيكي جداً في الرياضيات ومن أهم نتائج الطريقة الموضوعاتية (طريقة المسلمات).

لا شك في أن وجود ملاحظات مدونة جيدة، يشكل ميزة لجميع هذه العمليات التي تتطلب فهماً معمقاً بدلالات النظام وأطوار ارتقائه المحتملة، وهذا ليس فقط للتواصل، إنما للتفكير أيضاً، علماً بأن الملاحظة المدونة لا تغني أبداً عن الفكرة الواضحة.

الفصل السادس

البرمجة والاختبارات

تمثل البرمجة النشاط الرئيس والأكثر شهرة في دورة تطوير النظام البرمجي، فهي تجسد «روح» النظام. إلا أن هذه العملية لا يمكن اختزالها بوجود البرنامج وحده، لأنها تتضمن عدداً من الأنشطة الأخرى الضرورية لديمومة البرنامج. فالحديث إذن هو في سياق «البرنامج المنتج» عوضاً عن «البرنامج»، لدى اجتماع العناصر التالية:



فالبرنامج من دون توثيق واختبارات لا يكون كياناً صناعياً وتكون مدة حياته محدودة جداً.

أولاً : البرمجة

١- نموذج البرمجة.

آ- البرنامج = الخوارزميات + هياكل أو بنى البيانات.

يعد نشاط البرمجة نتيجة مباشرة لبنية أجهزة الحاسوب وفق تصور جون فون نيومان (John von Neumann) منذ حقبة الأربعينات.

ومن الأهمية بمكان هنا قراءة النصين التأسيسيين في هذا السياق:

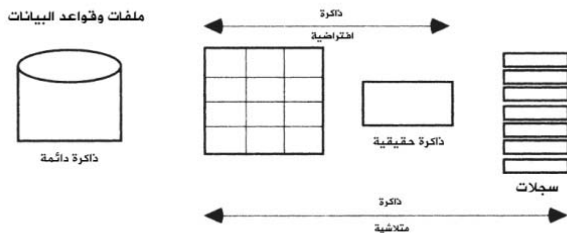
”نقاش مبدئي حول التصميم المنطقي لأداة حوسبة إلكترونية“
(“Preliminary discussion of the logical design of”)

(an electronic computing instrument Planning and coding) والترميز لأداة حوسبة إلكترونية“ (problems for an electronic computing instrument ، والتي تعود إلى عامي ١٩٤٦-١٩٤٧ (راجع ”الأعمال الكاملة“ ، المجلد ٥ ، Œuvres complètes. vol ٥). ففي نموذج نيومان، تشترك البيانات والتعليمات، والتي يشكل اجتماعها البرنامج، في ذاكرة واحدة ما يُمكن البرنامج من صنع برنامج آخر. وهذه السمة الأخيرة هي سمة محورية بكل معنى الكلمة.

فالبرمجة هي تنسيق البيانات والتعليمات معاً بهدف حل مسألة ما. ويكون الهدف موصفاً بواسطة سلسلة من التعليمات التي -إن نفذت على البيانات- قامت بتغيير هذه الأخيرة بشكل تدريجي إلى حين تحقيق الهدف. من هنا فإن البرنامج هو في الأساس استنتاج منطقي تكون فيه البيانات هي الأطراف، والقواعد المنطقية هي تعليمات الجهاز.

وبمجرد بلوغ البرنامج حجماً معيناً، ينبغي تنظيم هذه البيانات والتعليمات بمنتهى الدقة حتى تبقى واضحة ومفهومة. ومن مهام البرمجة، ضمان سمة الوضوح هذه من خلال تقديم آليات لترتيب وهيكلية هذه البيانات والتعليمات.

ويمثل الرسم التوضيحي التالي النمط الأساسي لنشاط البرمجة وسيرها بمرور الوقت:



الذاكرة الافتراضية وسيلة أُمدت بها أنظمة التشغيل منذ أوائل حقبة السبعينات، وساهمت بشكل كبير في تبسيط عملية البرمجة من خلال تزويد المبرمج بوهوم وجود ذاكرة حقيقية كبيرة الحجم. فهي وسيلة بارعة تنطوي على نقل شفرة من نوع (م) إلى شفرة من نوع (ث) إذ كان المبرمج في ذلك الحين هو الذي يدير الذاكرة التي كانت تعد نادرة نسبياً. إلا أنه لا يمكن إخفاء هذه الآليات الضمنية تماماً لأنها تتسبب أحياناً في تدهور أداء البرنامج من خلال جعل سلوكه غير حتمي (الأمر الذي يعد معوقاً وعقبة في حال الزمن الحقيقي). وتصبح السجلات خفية بمجرد استخدام لغة برمجة رفيعة المستوى، الأمر الذي يعد طريقة أخرى للانتقال من برنامج من نوع (م) إلى برنامج من نوع (ث). الجدير بالذكر أن المترجم إلى لغة الآلة يتولى كل ما يتعلق بتخصيص الموارد، وهو ما يمكننا من اختزال عدد التعليمات الأساسية إلى حد أدنى.

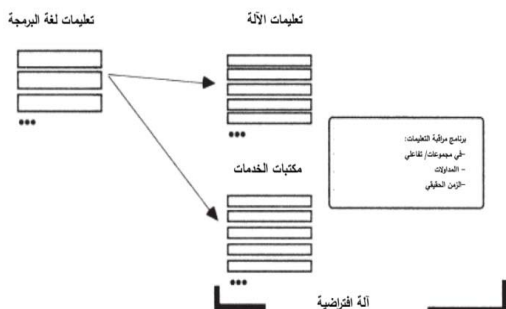
تمثل المجموعة المكونة من: (الذاكرة الافتراضية + الذاكرة الحقيقية + السجلات) حيز العنوان للبرنامج، إذ يمكن الرجوع إلى مكونات البرنامج عن طريق الاسم الذي اختاره المبرمج لكل منها، وذلك دون التدخل مع عنوان برنامج أخرى. ومن الوظائف الأساسية التي تقوم بها

مجموعة المترجم الى لغة الآلة، ومحرر الترابط ومُحَمِّل البرنامج إدارة حيز الأسماء هذا على نحو سليم.

توجد جميع تعليمات البرنامج في الذاكرة الافتراضية، ويمكن الإشارة إليها بواسطة الاسم الذي سيرتبط به العنوان المادي. إلا أن الأمر يختلف بالنسبة للبيانات الدائمة التي يتعذر إضافتها إلى حيز عنوان البرنامج، نظراً لحجمها الكبير واحتمال اشتراك برامج أخرى في استخدامها. ولا تُعرف هذه البيانات إلى بواسطة اسمها الرمزي، ولا يمكن العثور عليها إلا بواسطة وظائف نفاذ التي توفرها نظم إدارة قواعد البيانات ونظم الملفات. وفي حال لم تف هذه الوظائف بمتطلبات مهندس بانيان النظام أو المبرمج لأسباب تتعلق بالأداء على سبيل المثال، فإن من الممكن استبدالها بوظائف أنسب، وعندئذ لا يتم استخدام سوى الوظائف الأساسية الخاصة بنظام إدارة السجلات أو بالذاكرة الافتراضية للحصول على حيز ذاكرة في القرص. إلا أنه ينبغي التفكير ملياً في هذا الخيار قبل اعتماده، لأننا في أفضل الأحوال نكون قد استبدلنا برنامجاً من نوع بسيط (ث) بعدد كبير من شفرات أو رموز برنامج (م). وفي حال وجب في وقت لاحق اشتراك برامج أخرى في هذه البيانات، أصبح البرنامج من نوع (ب): واجتمعت بذلك كافة أسباب الحيد عن الحدود الزمنية/ التكلفة المرسومة.

كان تنظيم التعليمات موضع جهود حثيثة منذ بدايات الحواسيب الأولى (أول فورتران سنة ١٩٥٤)، إذ كان الهدف هو إخفاء تفاصيل آليات العتاد قدر المستطاع، والتي تكون بالغة التعقيد وتعسفية في الغالب حينما لا نكون على اطلاع بالقيود الخاصة بالعتاد. وفي بنية الآلات التي تُعرف بتقنية حاسب مجموعة التعليمات المعقدة CISC (أي معظم الأجهزة الحاسوبية الشائعة، مثل معالجات شركة إنتل الدقيقة)، كثيراً ما تنطوي الآلة على ما يناهز الـ ٣٠٠ تعليمات. من هنا فإن اختزال هذا العدد إلى بضع عشرات هو بمثابة اختزال للأفكار وعملية تبسيط عظيمة بالنسبة للمبرمجين، ما يمثل دوماً مصدر مكاسب جمة في سياق الإنتاجية. تقوم

إذن حصة تعليمات لغة البرمجة بتنظيم رؤية الآليات والخدمات الأساسية المتوافرة لتحويل البيانات الأساسية إلى بيانات ناتجة من جهة، كما تقوم من جهة أخرى بتمكين تسلسل العمليات بواسطة مراقب التعليمات الذي يدير النفاذ إلى الموارد. وفيما يلي رسم توضيحي يبين هذا المبدأ:



يتم ترتيب الخدمات في مكتبات يمكن النفاذ إليها في ظروف محددة، ويتم النفاذ إلى هذه الخدمات عن طريق وظائف تنفيذية متصلة بلغات البرمجة، مثل الوظائف الرياضية الخاصة بلغة فورتران، أو ما يعرف بـ «Report Writer» أي محرر التقرير الخاص بلغة كوبول.

- ويمكن تصنيف برامج مراقبة التعليمات في ثلاث مجموعات كبيرة:
- برامج المراقبة عن طريق المعالجة في مجموعات أو برامج مراقبة اقتسام الزمن، والتي تسمح بمعالجة العمليات على نحو تسلسلي؛
- برامج مراقبة المداولات والذي يتيح تقاسم مئآت أو آلاف المستخدمين مكتبة واحدة من البرامج، تعرف بالمداولات، مع ضمان سلامة البيانات وإيهام المستخدم بأنه وحده؛
- برامج مراقبة الزمن الحقيقي التي تسمح بإدارة الزمن الحقيقي للأحداث العشوائية التي تتطلب تنفيذ مهمة على وجه الأولوية،

وانقطاع وإعادة المعالجة.

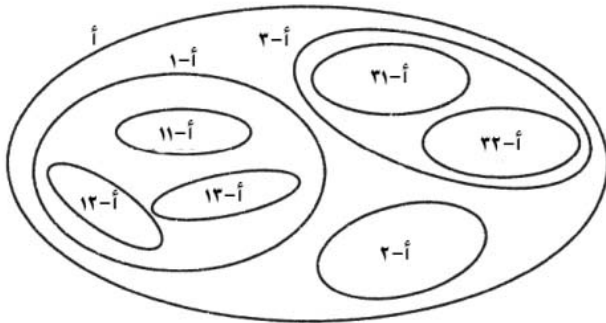
وتمثل هذه المجموعة ”الآلة الافتراضية“، التي تعد غاية توليد المترجمات إلى لغة الآلة.

تشكل رؤية هرم الذاكرة وتنظيم التعليمات نموذج البرمجة. ويتوقف هذا النموذج إلى حد كبير على لغة البرمجة، التي تعد مكوناً أساسياً، فيه لكنه بصفة عامة غير كاف للأسف، وتعد معرفة مكتملة قدر الإمكان لهذا النموذج شرطاً لا غنى عنها لكفاءة المبرمج.

من هنا فإن التحديات الأساسية هي في معرفة ما إذا كان النموذج منسقاً ومكتملاً: بمعنى توفر الإجابة، على كل سؤال قد يطرحه المبرمج على نفسه بشأن صحة وسلوك برنامجه، في النموذج فقط لا غير، وإلا اضطر إلى «النزول» إلى مستوى آليات الآلة الافتراضية. ويترتب عندئذ عليه معرفة الآلة الافتراضية إلى جانب قواعد تحول برنامجه في لغة هذه الآلة. وتتطوي مسألة الاتساق على معرفة ما إذا كانت تسفر صيغ متعددة ممكنة للبرنامج عن أوجه سلوك مقبول بغض النظر عن البيئة أو المحيط: وهنا تكمن إشكالية حتمية ودقة النموذج المطروح. وتعد هذه المسألة في حالة نظم الزمن الحقيقي ذات أهمية رئيسية، بل وتشكل عاملاً هاماً في اختيار لغة البرمجة والمراقب.

ب- تقديم بعض الآليات: تجري هيكلة حيز أسماء برنامج ما في مرحلتين:

- المرحلة ١: يكون عالم الأسماء منبسطاً ويمكن النفاذ إلى كل ما فيه من كل قبل، وهو نموذج لغات الجيل الأول فورتران وكوبول.
- المرحلة ٢: مع تعاظم حجم البرنامج وضرورة عمليات الترجمة المنفصلة، يزداد هذا العالم تعقيداً، وتنظم الأسماء في تسلسل هرمي:



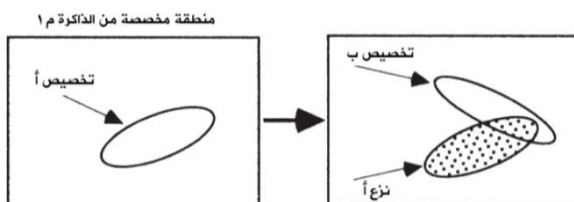
ترث الفقاعة الواحدة أسماء الفقاعات الحاوية لها، بيد أنها لا تعرف الأسماء التي لا توجد في خطها الهرمي. ويمكن لمبرمجي ١٢ وأ ١٣ أن يقوموا بتسمية عناصر البيانات وشفرة البرنامج كما يشاءون لأنها توجد في تسلسلات هرمية بيئية. وكان هذا المبدأ التنظيمي قد اعتمد للتعليمات الحسابية للغة فورتران، ولبينانات كويول.

وكانت أول لغة تمثل تناظراً تاماً هي لغة ألغول ٦٠، فقد كانت أول لغة محددة بشكل بالغ الدقة بيد أنها لم تلق رواجاً تجارياً. وسرعان ما لحقت بها لغة بي إل ١ (PL/1) التي ابتكرتها شركة أي بي إم، والتي حققت بعض النجاح، وكانت مصدر إلهام لعدد من اللغات بين ١٩٦٠-١٩٧٠ لا تزال قيد الاستخدام حتى اليوم. والجدير بالذكر أن كل ما نجد من هيكلة في لغات البرمجة الراهنة، مثل أدا وسي، ناجم بشكل مباشر عما كان في ألغول أو بي إل ١ وعلى نحو أفضل في بعض الأحيان.

وكان الابتكار الأخير في سياق لغات البرمجة هو مفهوم "النوع" الذي ظهر لأول مرة في لغة باسكال، وهي لغة برمجة ممتازة أخرى لم تلق قبولاً تجارياً.

ففي اللغات السابقة لباسكال، كانت الذاكرة توصف بواسطة أنواع معرفة بشكل مسبق خاصة بالآلة التابعة (البت، والحرف، والثابت العددي الصحيح الطويل، والثابت الصحيح القصير، والثابت العددي الحقيقي إلخ...). ويتميز هذا الأسلوب بتسهيل عمل المترجم إلى لغة الآلة، بيد أنه يعاني من كونه يخصص نفس التعريف لعدة كيانات متباينة منطقياً. أما النوع في لغة باسكال فيكون بمثابة معادلات للأبعاد تقوم بتعريف الأبعاد المادية، ويسمح النوع بالقيام بالعديد من الاختبارات والارتقاء بدقة دلالات التمثيل.

وفيما يتعلق بتنظيم الذاكرة، فقد انتقلنا من رؤية تتسم بالسكون، رؤية لا تتطوي سوى على قيم غير مركبة وجداول (فورتران، وكوبل التي كانت تتطوي بالإضافة إلى ذلك على بنى بسيطة)، إلى رؤية دينامية بشكل أكبر، كان للغة بي إل/ ١ أيضاً الدور المفصلي فيها. ففي نموذج بي إل/ ١، توجد مناطق من الذاكرة يمكن تخصيص أو نزع كيانات منها ذات أحجام متفاوتة.



ونتيجة لهذه الإدارة الدينامية، تصبح الأسماء «متغيرات»، قابلة للتحكم فيها، الأمر الذي يحقق بشكل كامل نموذج فون نيومان، وبخاصة حينما يكون نظام التشغيل يشمل محرر ترابط ديناميكي. فالكائن الموافق هو عنوان في المنطقة المخصصة من الذاكرة، ما ينشئ نوعاً جديداً (في

بي إل/١: pointer (مؤشر)، وفي أدا: Access (نفاذ). لكن إدارة هذه الكيانات الجديدة التي تزود النموذج بقوة فائقة، هي عملية بالغة الحساسية، فهناك احتمال الاحتفاظ بمؤشرات متجهة نحو مناطق أخليت، الأمر الذي يقوض اساق البرنامج، إذ أننا نصل إلى الكيان (ب) معتقدين أننا نصل إلى الكيان (أ)، وذلك على نحو غير حتمي. وفضلاً عن هذه الآلية الوظيفية بالغة الأهمية، ستكون هناك حاجة -كما هو الحال غالباً- لآلية تصحيحية متطورة إلى حد ما، للقيام باختبارات، تمكننا من نزع جميع المؤشرات الخاصة بالكيان أ لدى نزع أ. ومن هذه الآليات ما لا يمكن الاعتماد عليه إلا عندما تجري الآلة الكامنة بنفسها ببعض هذه الاختبارات، بواسطة مفاهيم مثل تقسيم أو تجزئة الذاكرة، وسجلات أساسية لا يمكن أن تحوي سوى عناوين، إلخ... وتوجد هذه الآليات في معظم الوحدات المركزية الراهنة، وكان أول استخدام لها في نظام مولتكس (Multics).

كانت ثمة محاولات عديدة، بالإضافة إلى الآليات الرئيسية، لدمج مفاهيم في لغات البرمجة محاذية لنظام التشغيل. وتعد المهام غير المتزامنة والاستثناءات مثلاً على ذلك.

أما الاستثناءات، فهي أحداث غير طبيعية بشكل عام يكشف عنها الجهاز، بيد أنه من المثير للاهتمام استرجاعها في البرنامج الذي طرأت فيه. فالحدث الاستثنائي وسيلة أساسية لتعزيز قدرة البرامج على التشخيص الذاتي ولتعزيز وثوقية البرنامج الحاسوبي. لكن هذه الآليات غير عملية في اعتمادها المفروض على الحاسوب وعلى نظامه التشغيلي.

تسمح المهام غير المتزامنة بتنظيم البرنامج على شكل مجموعة من العمليات المتعاونة، وينبغي عندئذ أن تتضمن لغة البرمجة نموذج عمليات (مثل نموذج سي إس بي (CSP) الذي وضعه العالم هور (HOARE) (تواصل العمليات المتعاقبة) والذي يستخدم بديلاً

له في لغة أدا) ينبغي ترجمته على صعيد البنى الكامنة في الحاسوب ونظامه التشغيلي. كما ينبغي أن تتضمن لغة البرمجة أساسيات إدارة السيرورات ووسيلة لمزامنة هذه السيرورات. الجدير بالذكر أن هذه الآليات تتيح تكيف سلوك البرنامج مع قيود البيئة التي يعمل فيها، وهنا تحديداً تكمن الصعوبة إذ ما من إجابة واحدة لتحد متغير بشكل واضح. ويختلف سلوك نظام المعاملات عن نظام الزمن الحقيقي. وبدلاً من إضافة مراقب للغة (كما هو الحال بالنسبة لأدا) ربما يكون من الأحكم ترتيب توافقية البرنامج لتتماشى مع أي نموذج لمراقب (وهو أسلوب لغة سي، وسي++ فضلاً عن العديد من اللغات السابقة لها، كما هو نهج كوبول مع نظم معالجة المداومات) وإن كان ثمة خطر بأن يتطلب الأمر بعض التضحيات، فالترجم إلى لغة الآلة في جميع الأحوال بعيد كل البعد عن القدرة على التحقق من كل الأمور، بيد أنه يسهل عليه تحديد المجالات التي تتسم بالصحة بشكل جزئي.

أسلوب البرمجة أمر هام بلا شك [١٩، ١٠] لكن البرمجة ليست ممارسة لأساليب شخصية، إنما هي عملية مفاضلة مستمرة بين سهولة القراءة والفعالية والسلامة إلخ.... أخيراً، لا بد من الإشارة إلى أهمية دور المترجم إلى لغة الآلة، الذي لولاه لكان نموذج البرمجة مجرد مفهوم نظري.

٢- عملية البرمجة :

تعود خصوصية عملية البرمجة بلا شك لكون هذا العمل ناجماً عن نشاط فرد متميز يضفي بصمته الخاصة عليه التي لا يمكن محوها أحياناً. وقد يكون للبرنامج عدة مؤلفين، إلا أنه لا يمكن أن يكون له أكثر من مؤلف واحد عند اللحظة (ز).

لحظة بدء عملية البرمجة، يوجد عدد من السطور يساوي الصفر، ويكون أمام المبرمج مجموعة من الملفات و/أو البرامج السابقة التي عليه

تغييرها وتجميعها لتكوين كيان جديد ينطوي بنهاية هذه العملية على عدد (ع) من السطور. وحتى ذلك الحين، سيكون عليه تحديد جميع البيانات الوسيطة وسلاسل البيانات اللازمة، وإدارة الموارد التي يتناولها البرنامج، وترتيب النص ليكون واضحاً وسهل القراءة (بما في ذلك بالنسبة له هو شخصياً)، وإضافة جميع المكونات الإضافية لضبطه وتطويره إلخ... وفي وقت لاحق، في مرحلة التكامل والصيانة، سيخضع البرنامج على الأرجح لتعديل متفاوت الأهمية، إما لتصحيحه، أو تغييره وإضافة وظائف جديدة إليه، وصولاً إلى دورة تطوير جديدة.

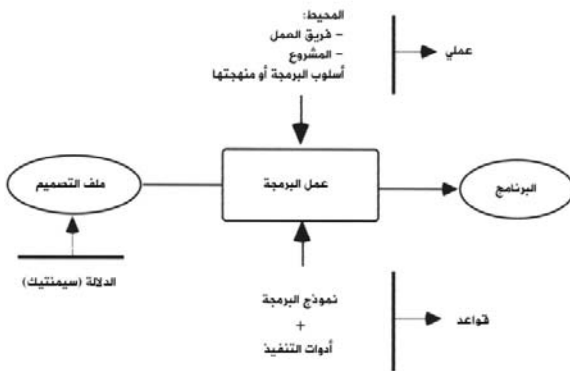
يتضح لنا إذن وجود جانبين مختلفين ينبغي النظر إليهما بعين الاعتبار:

- الجانب الراهن، أي النظر بعين الاعتبار إلى القيود القائمة لحظة الإنتاج؛

- الجانب التطوري، أي مراعاة القيود الموروثة من الماضي وتلك التي يمكن بشكل معقول التنبؤ بها مستقبلاً. فتاريخ البرنامج وأوجه تطوره أمر محوري بصفة عامة لتحديد عمره المتوقع.

وهذان الجانبان لا يتناغمان بالضرورة، فالقرارات التي تتخذ عند اللحظة (ز) حسب فهمنا للمسألة، ربما يعاد النظر فيها حينما يتسنى المزيد من الوقت لتعديل ما هو قائم. ومن التحديات التي تنطوي عليها البرمجة الحفاظ على تماسك البرنامج على الصعيد العام وإن وُجدت على الصعيد التفصيلي بعض مواطن الخلل الطفيف في هذا السياق.

آ- أوجه الوضع الراهن. - يمكن تمثيل عملية البرمجة من خلال المخطط البنيوي التالي:



البرنامج الناتج عبارة عن نص معد لتنفيذه على جهاز مستهدف؛ فهو رسالة بتعريف نظرية المعلومات وينبغي أن تحتوي عدداً من المعلومات الفائضة لضمان دقة ترجمة دلالة هذا النص البرمجي.

ب- الأوجه التطورية - أخذ ماضي ومستقبل البرنامج في الاعتبار يزيد بلا شك من تعقيد التمرين السابق. وتأتي مراحل البرنامج وفق التسلسل الزمني التالي:

- اختبارات الوحدات: البرنامج خاضع لسيطرة مؤلفه التامة.
- اختبارات التكامل: تم تسليم البرنامج إلى الفريق المسؤول عن تكامل مكونات البرنامج، فلم يعد خاضعاً لسيطرة المبرمج المباشرة، الذي لا زال يوسعه رغم ذلك تعديله مع اتباع احتياطات خاصة، أو إعادة تسليمه.
- التركيب أو التثبيت: تم تسليم البرنامج إلى فريق الدعم الفني، والتعديلات الوحيدة الممكنة هي المتعلقة بملف التوثيق الخاص بالعملاء.

- صيانة الإصلاح: تم تركيب البرنامج عند العملاء، والأفعال الوحيدة الممكنة هنا هي عمليات إعادة التسليم التي يقتضيها تصحيح البرنامج، فلم يعد للمؤلف أية سيطرة عليه. في حال الصيانة الارتقائية أو الهندسة العكسية، يتوجب على المبرمج استرداد بعض البرامج كما هي والتي ربما كان يود تعديلها، الأمر الذي قد يؤدي إلى خيارات ما كانت لتكون مبررة لولا هذا الشرط.

٣- إنتاجية البرمجة :

يطرح تحليل إنتاجية عملية البرمجة تحدي قياس البرمجيات بشكل عام، ولا تتوافر في الوقت الراهن أية وسيلة قياس خاصة بالبرنامج الحاسوبي يمكن مقارنتها بأي مما يستخدم في مجالات الهندسة الأخرى. والخوض بشكل مفضل في هذا السياق يتعدى نطاق هذا العمل [١١، ١٧]: إنما نكتفي هنا بعدد سطور شفرة البرنامج وبالمناطق السليم الذي لا ينبغي أن يفارق المهندس أبداً.

أول ما يتعين فعله هنا هو رسم حدود عملية البرمجة، وبخاصة فيما يتعلق بالاختبارات. ذلك أن مهمة البرمجة تُعد منجزة حين يكون قد:

١- كُتب البرنامج ؛

٢- أجريت جميع اختبارات الوحدات بنجاح وحققَت تغطية محددة مسبقاً لمخطط التحكم الخاص بالبرنامج؛

٣- مع توافر وثائق البرنامج.

نتناول حينئذ سطور البرنامج المختبرة والموثقة. وثمة نسبتان هنا تثيران الاهتمام بصفة خاصة:

٤- إجمالي الإنتاجية: عدد كسم/فع (٣٥٠ أمر لكل فع يعتبر معدلاً جيداً)؛

٥- معدل تقدم المهمة أو العملية: عدد كسم (مدة المهمة).

وبالرجوع إلى كل محور من محاور مخطط التزامن، نستطيع تحليل

أثر المحور على الإنتاجية، وتقدير الأهمية النسبية لكل من هذه المحاور.
 (أ) توثيق التصميم: تؤثر دقة التوثيق بشكل مزدوج على الإنتاجية:
 - من حيث حجم العمل الذي ينبغي القيام به: يتوقف حجم الخوارزميات إلى حد بعيد على كيفية تمثيل الكيانات؛
 - من حيث جودة المعلومات المقدمة: ينبغي أن يتسم توثيق التصميم بالاتساق والاكتمال فيما يتعلق بمنطق الوظائف التي ينبغي تنفيذها. كما ينبغي أن تزود ملفات التوثيق المعلومات اللازمة لتقدير حجم البرنامج، وبخاصة مناطق البيانات، وتثبت مستوى الموارد المتاحة للبرنامج.

وأي قصور في الدقة من شأنه أن يسفر عن:
 - إهدار الوقت عند البحث عن المعلومة المناسبة، فضلاً عن مواطن النسيان والأخطاء التي سيجرب تصحيحها أثناء الاختبارات.

(ب) نموذج البرمجة: يتجلى أثر نموذج البرمجة على صعيدين:
 - الصعيد الأول: اتساق النموذج واكتماله (أو التمام) المنطقي
 هي خصائص منطقية من حقنا توقع اتساق أي بيئة برمجية بها. وتتسم اللغات الحديثة مثل لغة أدا عادة بالاتساق، إلا أن هذا لم يكن الحال دوماً في السابق. إذ كانت تنطوي أولى خصائص اللغة سي على عدد من مواطن اللبس الضارة باستخدام هذه اللغة على نحو جيد. ويقوم المترجم إلى لغة الآلة، كملاذ أخير، بتسوية مواطن عدم التناسق، لكن ذلك يكون على حساب قدرة البرنامج على العمل في مختلف البيئات.

يشكل عدم الاكتمال معوقاً أكبر، فحين لا يكون النموذج مكتملاً، ينبغي العودة إلى الجهاز الكامن وإلى نظام التشغيل لضمان سمة الاكتمال هذه، كاستخدام نظام إدارة قاعدة البيانات أو برامج إدارة النوافذ مثل إكس/موتيف (x/MOTIF)، أو إدارة مراقب غير ذلك المدمج في هذه اللغة. ثمة نماذج متعددة إذن ربما لا تكون متسقة مع بعضها البعض،

وتضرر إلى حد بعيد بوضوح البرمجة والأهم من ذلك، سلوك البرنامج الذي قد ينحرف بشكل خطير. لغة أدا على سبيل المثال لا تنسجم قط مع نماذج أخرى، ما يشكل عائقاً كبيراً أمام انتشارها.

-الصعيد الثاني: تنفيذ النموذج على جهاز حقيقي.

هي في المقام الأول مشكلة المسافة التي تفصل نموذجين، فإذا تنامت عما يجب، ربما أصبحت عملية الترجمة، التي يعد المترجم إلى لغة الآلة المكون الأساس فيها، في غاية التعقيد؛ ويؤدي قصور في المعلومات إلى خيارات في عملية الترجمة ربما تحول البرنامج إلى كيان غير عملي. ولضمان التكافؤ على الصعيد الدلالي للبرنامج، سيقوم المترجم إلى لغة الآلة فيه أسوأ حال باستنتاج وتوليد برنامج بلغة الآلة فيه الكثير من الحشو أو الفائض أو إضافة مستوى تفسير وسيط لتقليص هذه المسافة (وهو حال اللغات الرمزية على غرار ليسب (LISP) أو برولوج (PROLOG)، ومعظم اللغات البرمجية غرضية التوجه).

الجدير بالذكر أن مما فاقم أوجه انتكاس لغات مثل بي إل / ١، وأدا مؤخراً، صعوبة تنفيذ مترجمين "جديدين" إلى لغة الآلة، فاللجوء إلى "مسكنات ألم" مثل تعليمات المترجم (PRAGMA) هو وسيلة مجردة تماماً من الأنافة وتقوم بإخفاء المشكلة، لأن البرامج لا تعود قابلة للعمل في بيئات مختلفة. كما أن ترجمة مفاهيم مثل تعدد الأوجه (المتغيرات أو الوظائف أو الكيانات) في اللغات البرمجية غرضية التوجه تتطلب آلية تحرير الروابط الديناميكية، التي تصبح عديمة الفعالية في حال لم يتم نظام التشغيل والعتاد بإدارتها؛ يرجى الاطلاع على الفقرة رقم ٧ أدناه بعنوان "البرمجة والمفاهيم غرضية التوجه".

إضافة إلى ذلك، فإن جميع النسخ الأولى من البرنامج ستكون مفعمة بالأخطاء (عند بداية عملية الاختبارات، كثيراً ما يصل معدل الأخطاء إلى ٥٠ خطأ/كسم)؛ من هنا فلا بد من إجراء القياس والمراقبة خلال مرحلة الاختبارات (التعقب، تحرير مناطق الذاكرة لجعلها في صيغة مفهومة، التحكم

بالتواتر). وهنا أيضاً، في حال تنامت المسافة عما يجب، باتت هذه العملية بالغة الصعوبة، وقد المبرمج جميع المزايا التي ظن أنه يكتسبها باستخدام لغة برمجية رفيعة المستوى، لن تحول للأسف قط دون ارتكاب الأخطاء.

(ج) البيئة الخارجية: يعد أثر المحيط الخارجي بالغ الأهمية بصفة عامة، ففريق العمل الذي يعاني من ضعف التواصل سيعمل بشكل غير فعال ويمضي وقته في تصحيح أخطائه الخاصة به. كما أن مشروعاً مداراً على نحو سيئ (ذا مهام موزعة بشكل غير جيد أو متوازية على نحو مفترط، وتقدير بخس لحجم العمل على نحو متكرر) من شأنه أن يقوض أفضل همم الأفراد، ويكون الحل الوحيد تغيير رئيس الفريق. كما أن تطبيقاً عقائدياً للمنهجيات دون مراعاة درجة النضوج الحقيقي للمنظمات يجعل الأفراد وفرق العمل غير منتجة.

(د) الأهمية النسبية لمختلف المحاور: نميل عادة إلى الاعتقاد بأن اللغة البرمجة أهمية بالغة، إلا أن الأمر في الواقع العملي ليس كذلك. ذلك أن أثر فريق العمل وخبرته وصعوبة المسألة أكثر أهمية بكثير. وما يصح في الرياضيات، من أن النظام الشكلي قد يساعد لكنه ليس في أي حال من الأحوال بديلاً للذكاء، يعد صحيحاً كذلك في عالم المعلوماتية. من الصعب حقيقةً تحديد ما إذا كانت لغة أدا متفوقة على لغة بي إل/ ١ بشكل موضوعي، أم بالعكس، فهي مسألة رأي بحث.

وفيما يتعلق بلغات البرمجة، لوحظ بالفعل وجود أوجه اختلاف ملموسة لدى الانتقال من «المجمعات/المجمعات الكلية»* إلى اللغات رفيعة المستوى ومن ثم إلى الجيل الرابع من لغات البرمجة (L4G) (حينما يتسنى ذلك).

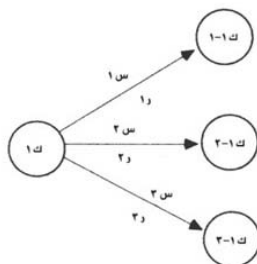
(هـ) دور الفرد: يكمن مفتاح الإنتاجية الحقيقي بيد المبرمج، فالمبرمج

* assembleurs / macro - assembleurs :

الجاد والمتمرس، ذو المعرفة الجيدة بالخوارزميات [٣]، والذي يحسن التصرف مع زملائه، قادر على تحقيق إنتاجية تزيد ١٠-٢٠ ضعفاً عن آخر لا يعرف سوى لغة البرمجة الخاصة به وذي علاقات مضطربة مع محيطه. فالتدريب والخبرة هما سبيل تحقيق الإنتاجية، الأمر الذي ينطبق على جميع مستويات المنظمة القائمة بالتطوير.

٤- تشييد كيانات رفيعة المستوى: وقفنا في الفصل السابق عند

أهمية الآلات ذات الحالات محدودة العدد، فهي أجهزة تسهل برمجتها. في حال رسم مخطط «حالة-الانتقال»، يمكن تمثيل البيان بتعليمات CASE:



في لغة برمجة من نوع أدا، هذا يؤدي إلى:

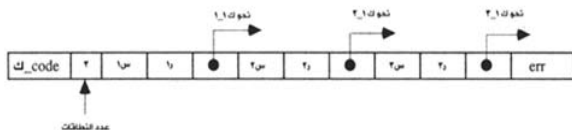
```
<< الحالة ١ ك > BEGIN العمل المتصل ب ١ ك END;
CASE SYMBOLE IS
WHEN ١ س => BEGIN العمل ١ ; GOTC الحالة ١-١ ك ; END;
WHEN ٢ س => BEGIN العمل ٢ ; GOTO الحالة ٢-١ ك ; END;
WHEN ٣ س => BEGIN العمل ٣ ; GOTO الحالة ٣-١ ك ; END;
WHEN OTHERS=> بمعالجة الأخطاء;
END CASE;
```

وذلك لكل عقدة من عقد المخطط.

في حال تكرر الأعمال في مختلف عقد البيان، يمكننا تعريف إجراء "عمل" ACTION يتضمن CASE مع WHEN لكل عمل، ما يحقق ميزة تحليل شفرة البرنامج إلى عوامل بشكل صحيح. عندئذ يصبح النمط الأساس للمخطط كما يلي:

```
<< الحالة ك > BEGIN ACTION_ ك (ك_CODE) : END;
CASE SYMOLE IS
WHEN ١ => BEGIN (العقد ١) ; GOTC ١_ك الحالة ; END;
WHEN ٢ => BEGIN (العقد ٢) ; GOTO ٢_ك الحالة ; END;
WHEN ٣ => BEGIN (العقد ٣) ; GOTO ٣_ك الحالة ; END;
WHEN OTHERS=> ERROR (err_Q1 ١ ك);
END CASE;
```

يظهر لنا نمط بالغ الانتظام بوسعنا ترميزه بغرض التبسيط في حال تعاظم حجم المخطط، وهذا النمط هو تعليمة آلة مجردة يمكننا تمثيله كما يلي:



بذلك نكون قد حولنا بنية تحكم وضبط ربما تكون بالغة التعقيد، إلى "جدول" سنتمكن من التحكم فيه وترميزه وتأويله بما يتفق والدلالة التي نود أن ترتبط بها. وهذا يدل مجدداً على مبدأ الازدواجية بين الأوامر التعليمية والبيانات، وهو جوهر نموذج فون نيومان. ثمة علاقة تقابل بين هذين الشكلين، ووحدها مقومات الملاءمة (التضام وسهولة الضبط

والمحاكاة إلخ...) هي التي تملي أسلوب تمثيل الذاكرة الأفضل.

٥- تحديد متغيرات* البرنامج: إن تحديد المتغيرات جانب

جوهرى من عملية البرمجة. وينطوي ذلك على القدرة على تجديد برنامج سليم بعد إضفاء بعض التعديلات. وتتناول هذه العملية البيانات و/أو شفرة البرنامج.

الجدير بالذكر أن فكرة تحديد متغيرات البرنامج في حد ذاتها متأصلة في البرمجة بسبب أوجه التعميم التي تنطوي عليها هذه العملية. إذ ينجم كل برنامج حقيقي عن خيار كان لا بد من اعتماده، إلا أن هذه الخيارات كانت اختلفت لو اختلفت المتطلبات والتصميم. كما أن ثمة احتمال لأي إضافة للبرنامج، وإن لم تلب حاجة محددة، أن تُطلب من قبل نسخة لاحقة من البرنامج. تنطوي هذه العملية إذن على إجراء اللازم لتسهيل تعديل البرنامج عندما يحين موعد التعديل. هذه القدرة على «استشعار» ما قد يتطلب التعديل، والتصرف بناء على ذلك، هو سمة نفسية مميزة للمبرمجين الجيدين.

ويمكن تمييز بعض أكثر أسباب التعديل انتشاراً:

- التغليف (أو الكبسلة): وهو حال مجموعة الرموز التي تمكن من احتواء جهاز أو برنامج لم يؤلفه المبرمج؛
- أوجه التصحيح: ويمكن لها أن تطال البيانات و/أو التعليمات؛
- وتتلخص القاعدة في تخصيص اسم رمزي لكل ما هو معرض للتحرك ويتطلب وصفاً أوحداً. وتعد مسألة إدارة الأسماء بشكل عام بالغة التعقيد بالنسبة للبرامج كبيرة الحجم. وتتيح معظم لغات البرمجة عدداً من التسهيلات للقيام بهذا النوع من التغيير مثل:

* متغيرات = parameters باللغة الانجليزية:

-الصيانة التكيفية: بمعنى إثراء هيكل البيانات و/أو بنية ضبط البرنامج.

ولادخال "المتغيرات" ثمة آليات مختلفة في تناول المبرمج:

- تلك التي تنطوي عليها لغة البرمجة والتي ينفذها المترجم إلى اللغة الآلة (وهي باللغة التطور في لغة أدا فيما يتعلق بالحزم، والوحدات العامة، والحمل الزائد، إلخ...):

- تلك التي تقتضي استخدام معالج إضافي مثل مولد الماكرو الخاص بلغة البرمجة سي.

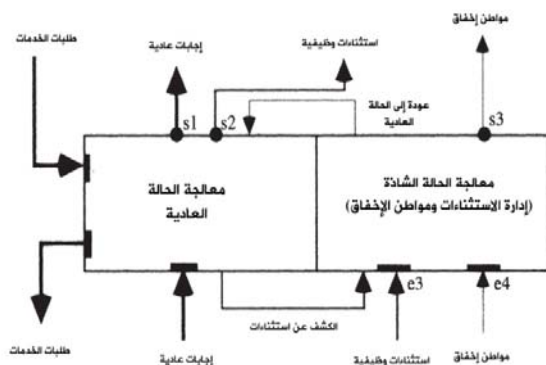
تواجهنا هنا مجدداً مسألة الاكتمال. فهل ينبغي أن تتضمن لغة البرمجة الآليات الخاصة بإدخال «المتغيرات» الممكنة، وهو أمر محال طبعاً، أو أنه ينبغي أن تنحصر هذه الآليات بالضرورة البحتة (على سبيل المثال المتغيرات الخاصة بتخصيص البيانات، والترميز الخاص بالنوع المقطع، إلخ...)؟ ففي جميع الأحوال ستكون ثمة حاجة للجوء إلى معالج إضافي أو أكثر.

نظراً لسمّة البساطة الضرورية التي لا ينبغي أن يتخلى عنها أي مصنع أدوات، يفضل تجنب إعاقه لغة البرمجة بآليات كثيراً ما تكون بلا جدوى (مثل فرض الحمل الزائد على المشغلين ما يجعل البرنامج صعب القراءة) فهي تعود على كل حال إلى الدلالة العامة للبرنامج وليس اللغة. ومن أكثر آليات الإضافة المستخدمة هي مولدات الماكرو، التي تمكننا من القيام بمختلف أنواع التحكم الرمزي بتسلسلات من حروف برنامج؛ فهي أدوات لا غنى عنها. وربما تكون مرتبطة بشكل غير مباشر بلغة تعرف قواعدها، كما قد تكون محايدة تماماً.

وقد يسفر تحديد المتغيرات بشكل صحيح عن مكاسب عظيمة في الإنتاجية.

٦- معالجة الأخطاء والضبط الذاتي: يتعلق هذا الجزء من البرنامج بالأجزاء ٢ و ٢ من النمط الأساس لنشاط البرمجة (راجع الفقرة ١ ، نموذج البرمجة): فهو يساهم في التحقق الديناميكي من تنفيذ البرنامج وفي تعزيز قوته. وبالنظر إلى الضرورة المتنامية لتحقيق الوثوقية في البرنامج الحاسوبي، بات هذا الجانب من عملية البرمجة أكثر فأكثر أهمية. ونعرض فيما يلي المبدأ الخاص بها.

يمكن تمثيل برنامج مثالي كما يلي:



ثمة أربعة منافذ لدخول البيانات برنامجاً من هذا القبيل من ، (e١) إلى (e٤) يتطلب كل منها نوعاً خاصاً من المعالجة:

- طلبات الخدمات؛ وهو المنفذ الطبيعي الذي يمكن من خلاله التحقق من بيانات الدخول والمصادقة عليها؛
- الإجابات العادية؛ وهي الإجابات التي تقدمها الخدمات المطلوبة والتي ينبغي التحقق من صحتها والمصادقة عليها.
- الاستثناءات الوظيفية؛ يتلقى البرنامج استثناءً بوسعه معالجته (مثل

تحرير أحد الموارد على سبيل المثال).

- مواطن الإخفاق؛ يتلقى البرنامج استثناءً يظهر تدهور النظام. ينبغي أن تكون التعليمات الخاصة بمعالجة مثل هذه الحالة محمية خاصة.

لا بد من الفصل تماماً بين الاستثناءات الوظيفية وبين مواطن الإخفاق.

وتوجد ثلاث نقاط خروج (S_1 إلى S_3)

- الإجابات العادية؛

- الاستثناءات الوظيفية؛

- مواطن الإخفاق.

ولكل منها دلالة متميزة.

جميع البرامج الخاصة بمعالجة الاستثناءات ومواطن الإخفاق هي من النوع (ب) وبالغة الحساسية للبيئة الخارجية. هذا وتعرض عملية استرجاع الأحداث الموافقة الآليات المقترحة من قبل لغة البرمجة، وآليات مراقب المعالجة، وآليات نظام التشغيل الضمني، وآليات العتاد في آن واحد للخطر. لذا فمن الضروري هنا السيطرة بشكل كامل على تعايش مختلف هذه الآليات عندما يكون ثمة احتمال برمجة هذا النوع من الواجهات.

كما ينبغي أن تراعى عمليات المعالجة التي ينبغي إجراؤها احتمال كون هذه البيانات تالفة، وعدم توافر بعض الموارد، وزوال موثوقية البيئة. لذا لا بد وأن تكون الشفرة المقابلة أكثر مكنة من الشفرة التي يضمن عملها على نحو سليم. ولا يمكن أن تُجرى الاختبارات الموافقة إلا بواسطة محاكاة الأعطال، وينبغي أن يكون حجم هذه الشفرة مضبوطاً ضبطاً تاماً ولا يكلف بالمهمة الموافقة لهذه الشفرة سوى أحد المبرمجين المتمرسين.

وبالنظر إلى ارتقاء النظم، باتت عمليات المعالجة متنامية الأهمية بيد أنها باهظة الثمن؛ وقد تحط في حال عدم انسجامها التام مع الشفرة الإسمية من أوجه الأداء إلى حد كبير، دون الآثار المتوقعة في حال حدوث مشكلات.

٧- البرمجة والمفاهيم غرضية التوجه : البرمجة غرضية

التوجه ليس بفكرة جديدة، فقد ظهرت قرب أواخر حقبة الستينات وبداية السبعينات في مجالين متميزين من مجالات التطبيقات: المحاكاة مع لغة سيمولا (Simula) عام ١٩٦٧، ومجال الواجهات الرسومية مع سمولتوك (Smalltalk) عام ١٩٧٢. وقد كان لمفهوم "الغرض" دور كبير في تعريف لغة أدا قرب نهاية السبعينات (إذ كان الحديث عن التغليف وأنواع البيانات المجردة بشكل أكبر). وتعد الأساليب غرضية التوجه أكثر حداثة ويمكن اعتبارها تكملة طبيعية للبرمجة الغرضية ذات المراحل المبكرة من دورة التطوير؛ إذ تدعمها مختلف اللغات، لعل أشهرها لغة النمذجة الموحدة التي اقترحتها مجموعة إدارة الكيانات (UML of OMG) والتي تدمج طيفاً من الجهود السابقة (موريز، منهج التصميم والتحليل البنوي / SART/SADT، مخططات انتقالية الحالة، ومنهج النمذجة الغرضية ومخططات الفئات التي هي في الواقع الرسوم التخطيطية للعلاقات - الكيانات الأساس لمنهجية موريز (MERISE*)، إلخ...).

هذا وترمي البرمجة الغرضية والمنهجيات المصاحبة لها، إلى توحيد طيف الآليات القائمة منذ زمن بعيد في بيئات الترجمة و/أو أدوات الهندسة البرمجية بمساعدة الحاسوب، وذلك وفق ثلاثة مبادئ رئيسية: الفئنة، والكائن، والإرث، وبعض المبادئ الأخرى المنحدرة منها مثل تحديد الأنواع والحوسبة العمومية وتعدد الأشكال أو الأوجه، إلخ...

ترمي عملية التوحيد هذه إلى تسهيل بناء المكونات البرمجية، وتعزيز هيكلتها في مكتبات من "الأغراض" التي يمكن إعادة استخدامها بشكل ساكن و/أو ديناميكي، وذلك بهدف تسهيل تنظيم البرمجة الهائلة اللازمة للمشاريع الضخمة. إلا أن السمة الأبرز للبرمجة غرضية التوجه

* Merise هي منهجية نمذجة تستخدم لأغراض عامة في مجال تطوير نظم المعلومات وهندسة البرمجيات وإدارة المشاريع :

هي تلاشي الفصل التقليدي بين الجزء الخاص بالبيانات والجزء الخاص بالتعليمات من البرامج، والذي يعد الفصل الصارم بين قسم البيانات وقسم الإجراءات (Data division, procedure division) في لغة كويول أفضل مثال عليه. إذ أنه يُنظر هنا إلى البيانات والعمليات كبنى ثنائية. وهذا ليس خطأ في نهاية المطاف، وإن أوجد ذلك أحياناً صعوبات في مرحلة التعلم، لأننا كثيراً ما ندرك المجموعات الساكنة (أي البيانات المخزونة في الذاكرة) قبل التغيرات الديناميكية (التعليمات التي تتحكم بهذه البيانات). ويستند مبدأ الهيكل الجديد هذا إلى مفهوم الأنواع المجردة من البيانات (إيه دي تي) ADT الذي سبق بكثير ظهور اللغات الغرضية.

ويعرف الفئة بما فيه من العمليات التي تسمى "طرقاً" في البرمجة الغرضية نوعاً مجرداً من البيانات. ويعد الغرض نقطة معينة من فئة ما أثناء البناء الساكن أو الديناميكي للبرنامج الحاسوبي. ترتيب الفئات بشكل هرمي يسمح بتحديد قواعد الإرث. ويمكن الاطلاع على مرجع جيد خاص ببرمجة وتصميم الغرض في المرجع [١٩].

ثانياً: الاختبارات

١- طبيعة الاختبارات وغايتها.

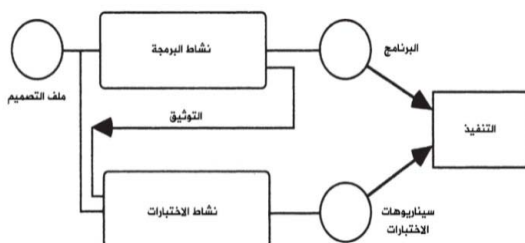
(أ) تعريفات وأهداف: تمثل الاختبارات وسيلة للتأكد من صحة ما قمنا ببرمجته. بمعنى آخر، في حال زدنا البرنامج ببيانات تحترم بروتوكول النداء، أخرج البرنامج نتائج متوقعة في الوقت المخصص له ودون استهلاك موارد أكثر من تلك المتاحة له، وإلا قمنا بتشخيص خطأ وتقصى سببه.

ولا ينفصل نشاط الاختبار عن نشاط البرمجة، الذي يمثل ثنائياً معه. يقتضي كل نشاط مجهوداً معيناً (مج). إلا أنه ينبغي تعظيم فعالية

المجموع التالي قدر الإمكان:

مج (إجمالي) = مج (برمجة) + مج (اختبار).

يفرض الاختبار حكماً تنفيذ البرنامج، بيد أن هذه ليست الطريقة الوحيدة لتصحيح برنامج ما. إذ يمكن إيجاد أخطاء عديدة بعيداً عن تنفيذ البرنامج، عن طريق مراجعة التصميم، وفحص الشفرة، وإعادة القراءة بشكل متقاطع، والتجارب، إلخ..



وهنا أيضاً يكون الجهد المخصص لاكتشاف الأخطاء مجموعاً

يساوي: مج (الخطأ) = مج (مراجعات) + مج (اختبار)؛

وهو المجموع الذي ينبغي تعظيم فعاليته.

ويعد الاختبار بروتوكول تجريبي بالغ الدقة ويمكن تكراره، من شأنه أن يثير طيفاً من الإجابات، التي تبين سلامة عملية التغيير التي أجراها البرنامج. صياغة هذه الاختبارات عملية مكلفة، فضلاً عن كون الحالات الاندماجية المحتملة الناجمة عن بنية البيانات الواردة، ونهج العمل المجازة، ومختلف الإعدادات، مزيجاً خطراً يصعب التحكم فيه.

وفي نظام برمجي ذي عدد (م) من المداخل وعدد (ن) من المخارج ذات س₁، س₂...، س_م وص₁، ص₂...، ص_ن من السيناريوهات على التوالي، يكون عدد السيناريوهات المحتملة:

(ص ١ x ص ٢... x ص ن) $x_1, x_2, \dots, x_n$ تكون معظمها حالات رفض. وأمام هذا العدد الهائل، يكمن التحدي في إيجاد اختبارات «جيدة»، أي اختبارات من شأنها إيجاد الأخطاء التي لا نعرف عددها ولا موطئها.

ب- البيانات الإحصائية: أجريت دراسات عديدة بشأن توزيع وتكلفة الأخطاء [١٦]. وتظهر هذه الدراسات بصفة عامة:

- أن ثلثي الأخطاء ينجم عن أخطاء في الدراسة والتحليل والتصميم؛

- أن ثلث الأخطاء ينجم عن البرمجة.

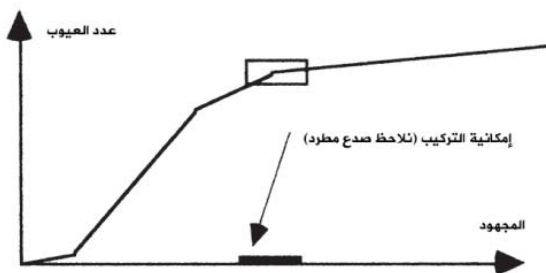
هذا وتتنامى تكلفة الأخطاء كلما تباعدت المسافة الزمنية التي تفصل المرحلة التي أحدثت فيها هذه الأخطاء عن المرحلة التي تم فيها الكشف عن هذه الأخطاء. ويمكن وضع الجدول التالي الذي يوضح بعض المعدلات:

التحديد التقني للمتطلبات، والتصميم العام	١			
التصميم	٢,٥	١		
البرمجة	٦	٢,٥	١	
التكامل	٢٥	١٠	٤	١
التشغيل	٧٥	٣٠	١٢,٥	٣

سيكلف خطأ في التعبير عن المتطلبات إن تم تصحيحه في مرحلة التشغيل ٧٥ ضعف ما سيكلفه إن تم الكشف عنه خلال مرحلة التعبير عن المتطلبات، ومن هنا تتبع أهمية النماذج البدئية والتجريبية.

وحتى بلوغ مرحلة البرمجة، لا يمكننا إجراء الاختبارات، بيد أن سياسة كشف مبكر عن الأخطاء بمساعدة وسائل المراجعة ستكون مثمرة بلا شك.

ج - ديناميكية المجهود الخاص بالاختبارات: ما يهمنا هنا هو عدد العيوب التي يتم الكشف عنها وفق المجهود المبذول في مرحلة الاختبارات. وهذه الديناميكية هي أيضا على شكل منحنى S.



وللتكلفة المتوسطة لكشف-تصحيح خلل ما الشكل القطعي المكافئ
التالي (استمداد من المنحنى S ومتعلق بمشتقه).

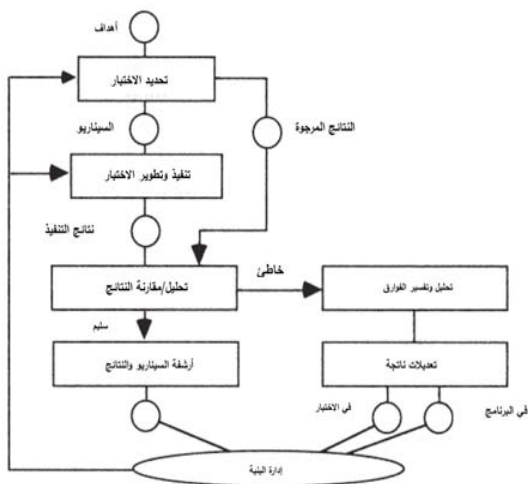


في مرحلة ما، يفضل إذا تثبتت أو تركيب البرنامج الحاسوبي، لأن
تكلفة مواطن الخلل التي يتم الكشف عنها بعد تركيبه في "المنصة"
ستكون أقل منها في حال لزم صنع سيناريوهات الاختبارات الموافقة. أما
بالنسبة لتكلفة التصحيح بعد التركيب في المنصة، فينبغي إدراج تكلفة

نتائج العطل، وبخاصة بالنسبة للبرمجيات الجاهزة للاستخدام (مفتاح باليد).

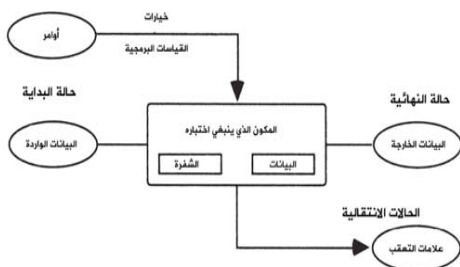
النتيجة الظاهرة لمجهود مرحلة الاختبارات هي سلسلة من سيناريوهات الاختبارات كشف البعض منها عن أخطاء، ولتكن: ت_١، ت_٢،...، ت_{١٠}. فلنفترض أن ت_١ كشف عن خطأ ما، ما أسفر عن تعديل البرنامج. لتأكد من إحرازنا تقدم فعلاً، علينا إعادة تنفيذ كل أو جزء من سلسلة ت_١، ت_٢،...، ت_{١٠} إضافة إلى ت_١. ثمة تباطؤ لا مفر منه إذن يتنامى بقدر عدم ملاءمة أو غياب أدوات إدارة الاختبارات. ويطابق هذا التباطؤ الطرف في المعادلة 1 ت الميزة للمنحنى S (انظر الفصل الرابع الفقرة (١)).

د- دورة تطوير الاختبارات: تجري صياغة الاختبارات وفق الأهداف الناجمة عن الاستراتيجية المتبعة والمعلومات المتوافرة. ويتطلب تحليل النتائج، الذي يكون عادة عبارة عن مزيج من الوسائل الاستنتاجية والاستقرائية، قدرات منطقية عالية، وبخاصة لدى صياغة النظريات التفسيرية التي ينبغي النظر إليها على أنها نظريات مصغرة تشكل إطار التعديلات التي يتم إجراؤها. ولا تقل صعوبة كتابة اختبارات جيدة عن صعوبة كتابة برامج جيدة. وكثيراً ما يُقلل من صعوبة الاختبارات، وكثيراً ما يتم تكليف موظفين ذوي مؤهلات أقل بالمهام الخاصة بها؛ وحينئذ يكون الأداء سيئاً جداً. وفيما يلي دورة تطوير الاختبارات:



وكلما بعدنا عن هذا المخطط، قل اعتبار عملنا بعمل الاختبار.

٢- اختبارات «الصندوق الأسود» واختبارات «الصندوق الأبيض».



يمكن تمثيل رؤيتنا لأحد مكونات البرنامج الحاسوبي الذي ينبغي اختباره بالمخطط التالي:

تحدد مواصفات البرنامج الحاليين المبدئية والنهائية بشكل جيد ، في حين تتوقف الحالات الانتقالية على دقة الملاحظة المطلوبة: فقد تكون بالغة الكبر وتغير من سلوك البرنامج.

أ- تعريف ودور اختبارات «الصندوق الأسود»: لا ننظر بعين الاعتبار هنا سوى إلى البيانات الواردة والبيانات الخارجة ، مع حظر النظر إلى مضمون المكون.

هي عملية المصادقة على البرنامج الحاسوبي قياساً بمواصفاته. ينبغي إذن دراسة البنية الدقيقة لهذه النطاقات والمجموعات الموافقة لها. ولتجنب مواجهة معضلة الحالات هائلة العدد المحتملة التي أشرنا إليها أعلاه ، ثمة حاجة لاستراتيجية تطوير اختبارات المصادقة قائمة على معرفة عميقة لنهج استخدام البرنامج الحاسوبي. وانطلاقاً من مجموعة من السيناريوهات التي تمثل قدر المستطاع متطلبات المستخدمين ، نستطيع تحديد المتغيرات للتحقق من قوة البرنامج وأوجه أدائه إلخ...

ينطوي الجانب غير المناسب لاختبارات الصندوق الأسود أولاً عن الحالات التبادلية كثيرة العدد التي ربما تنتج عائداً متديناً للجهد المبذول في الاختبار ، وبخاصة حينما لا تمثل الاختبارات الاستخدام الحقيقي للبرنامج الحاسوبي ، أو عندما لا يكون المكلفون بعملية التأهيل على اطلاع جيد بالموضوع. بل إن تحليل النطاقات قد يكون شبه محال عند افتقارنا لحد أدنى من المعلومات الخاصة ببنية البرنامج الحاسوبي ، الأمر الذي يحول بدوره دون التحقق من اجتياز الحدود.

ب- تعريف ودور اختبارات «الصندوق الأبيض»: ننظر هنا بعين الاعتبار إلى المكون الذي نقوم باختباره بأكمله ، ونراقب المكون بدرجة ما

من التفصيل بمساعدة أدوات القياس المتوافرة في البرنامج و/أو نظام التشغيل. هي عملية التحقق من صحة عمل البرنامج الحاسوبي من حيث تنفيذه على الصعيد العملي.

تلقي اختبارات الصندوق الأبيض نظرة تحليلية متفاوتة في الدقة على سلوك مكون البرنامج. وهنا تحديداً تكمن الصعوبة، فحجم المعلومات التي يتم تناولها قد يتعاظم، حسب درجة تفصيل هذه النظرة. وتعتمد الاختبارات كلياً على بنية المكون، وبخاصة عندما تكون النظرة التي نلقيها دقيقة. وقد تكون بيئة الاختبار بالغة التعقيد، وبخاصة حينما ينبغي محاكاة مكونات غير متكاملة.

إن التحقق بواسطة اختبارات المصادقة ليس عملية سهلة بشكل عام، وقد يسفر عن وضع لا نعرف فيه على وجه التحديد ما الذي تم التحقق منه من خلال عملية المصادقة. وينبغي تنفيذ اختبارات سلوك البرنامج بحد أدنى من أدوات القياس، بحيث يتعرض السلوك قيد الملاحظة لحد أدنى من الإزعاج حتى يعتبر ممثلاً بالقدر الكافي.

وتتطوي استراتيجيات اختبارات جيدة على جمع اختبارات الصندوق

الأسود و اختبارات الصندوق الأبيض مع تنويع وجهات النظر:

- بنيان النظام، حيث نهتم بصفة خاصة بالواجهات البينية؛
- البرمجة، التي ننظر فيها في سياق تغطية تعليمات البرنامج؛
- التكامل، حيث نقف عند تتابع الوظائف بفرض تكوين سلاسل طويلة بحيث نستطيع تقدير المدة الوسطية قبل حدوث العطل (Mean time to fail) للبرنامج الحاسوبي؛
- الصيانة، التي نقوم فيها بحساب المدة الوسطية قبل الإصلاح (Mean time to repair)؛

- التشغيل، حيث نقف عند السلوك العام ومنظور المستخدم الذي نقوم بوصفه بواسطة زمن الاستجابة والمدة الوسطية بين مقاطعات النظام (Mean time between system interrupt)

للبرنامج الحاسوبي.

لا بد إذن من مقاييس جيدة لتجنب هدر الجهد في هذه المرحلة.

٣- تحليل المسارات: يخرج تفصيل عملية التحليل عن نطاق

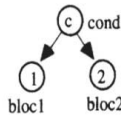
هذا الكتاب [١٦]، بيد أنها من الأهمية بحيث تستدعي رسم خطوطها الرئيسة.

بوسعنا دوماً ربط كل برنامج بمخطط ينمذج تعاقب التعليمات فيه: وهو ما يعرف بالهيكل التنظيمي للبرنامج*، إلا أنه ناتج هذا البرنامج وليس مخطط مواصفاته. هذا ويعالج مترجم اللغة رقيقة المستوى، الذي يحول البرنامج إلى لغة الآلة، بيان التحكم لتوليد سطور لغة الآلة وتحسين فعاليتها قدر الإمكان. كما يسهل الاستخدام المنظوم لأسس البرمجة البنيوية صياغة بيان التحكم ويزود المخطط بخصائص مثيرة للاهتمام (المخططات الشجرية للمناطق المتراكبة). ويمكن إظهار الرسوم البيانية ذات الصلة كما يلي:

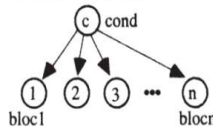
BEGIN
<bloc>
END



IF <cond>
THEN <bloc1>
ELSE <bloc2>

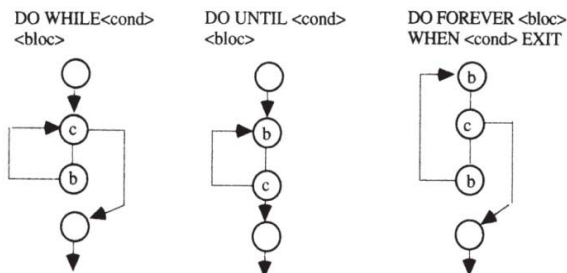


CASE OF <cond>
WHEN <bloc1>
WHEN <bloc2>
...
DEFAULT <blocn>



ويمكن تمثيل مختلف أنواع الحلقات كما يلي:

* organigramme :



وانطلاقاً من هذه الرسوم البيانية التي تعرفها المترجمات إلى لغة الآلة، يمكننا تعريف عدد من العمليات بالغة البساطة والفائدة.

فحين يتم تنفيذ البرنامج مع مجموعة معينة من البيانات، ينجم مسار ما على الرسم البياني، ويشكل مجموع المسارات الناجمة "تغطية". ومن هذه المسارات ما يسهل جداً التحقق من صحته:

- تغطية جميع العقد ويتضمن هذا الاختبار إدخال مجموعات من البيانات تمكننا من التأكد من تنفيذ جميع العقد لمرة واحدة على الأقل. وتعرف هذه التغطية المبدئية بـ "ت".

- تغطية جميع الأقواس: أي إدخال مجموعات من البيانات تسمح بالتأكد من المرور بجميع أقواس الرسم البياني لمرة واحدة على الأقل. وتعرف هذه التغطية بـ "ت".

ومن السهل جداً التأكد من هاتين التغطيتين:

- إما من خلال معالم التعقب التي وضعها المبرمج وبمساعدة أدوات قياسات البرمجيات.

- وإما باستعمال محلل برنامج المصدر، ويوجد من هذا المحلل عدة أنواع في السوق.

- وإما بواسطة المترجم إلى لغة الآلة، وهو الوسيلة الأفضل على

الإطلاق، حيث يعالج بالضرورة هذا البيان، على مستوى لغة الآلة و/أو على صعيد بنية وحدة البرنامج. هذا ويكون عائد الجهد المبذول على الاختبارات انطلافاً من هذه التغطيات، ممتازاً، وبخاصة لدى ربطات ١ باستخدام منظوم للبرمجة البنيوية، التي تثبت من خلال ذلك جدواها الاقتصادية الحقيقية، وليس فقط قيمتها الجمالية.

إلا أن هذه التغطيات لا تستنفذ للأسف الحالات الكثيرة المحتملة للبرنامج، بيد أنها تشكل مرجعاً لا غنى عنه. ونلاحظ على سبيل المثال في الحلقات أعلاه أن بوسعنا الحصول على تغطية ت ١ للحلقة بإجراء دورتين في الحلقة، إلا أن الاختبارات المثيرة للاهتمام هي:

- تنفيذ <وحدة>* الحلقة مرة واحدة على الأقل:

- عدم تنفيذ <الوحدة>

وذلك بالرغم من أن ت ١ لا يؤدي سوى إلى صنع اختبار واحد. وللخوض بشكل أكبر في استكشاف مجموعة الحالات الكثيرة المحتملة، ينبغي ربط بيان التحكم بآلة ذات حالات محدودة العدد نستطيع تحليل اللغة الخاصة بها (أي علامات التعقب التي تخلفها عملية التنفيذ). وكما الحال بالنسبة للرسوم البيانية الخاصة بعلاقات ارتباط البيانات، فإن هذه الدراسة خارجة عن سياق هذا المؤلف، إلا أنها تشكل الأساس النظري الوحيد الذي نستطيع به سبر أعماق التباديل الكثيرة بشكل منظم ومنضبط.

٤- اختبارات على الخط (الاختبارات الآتية). - يوصف

الاختبار بأنه «على الخط» حين يكون مدرجاً في البرنامج الحاسوبي، ونستطيع تفعيله حينما تطرأ أحداث معينة.

* (وحدة) bloc = :

ومن شأن مثل هذه الاختبارات أن تتيح التحقق من:

- البيانات الواردة إلى برنامج قبل تشغيله؛
- التماسك والاتساق البنيوي لمنطقة الذاكرة التي تتقاسمها برامج مختلفة؛

- تماسك الدلالات الخاصة ببعض الأحجام ذات الصلة بالبرنامج، سواء كان على صعيد البيانات (الارتباطات الوظيفية) أو السلوك (زمن الاستجابة، طول الطابور، عمق المكسد، إلخ).

في حال البرنامج الحاسوبي الذي تسود فيه برامج من نوع ب، تساهم هذه الاختبارات بشكل كبير في تعزيز قوة البرنامج، من خلال تسهيل الكشف عن الحالات التي يحتمل أن تكون شاذة قبل وقوع الخلل أو العطل. فلنعرض مثلاً على ذلك:

لنفترض أننا نسعى لتنفيذ العملية:

$$z = x \times y \quad \text{حيث} \quad x, y, z \in [0:65000]$$

و x, y , وراد من البيئة

حتى لا يكون z خارجاً عن طاقة البرنامج ينبغي التأكد مسبقاً من أن:

$$x \times y \leq 65000$$

ما يقودنا إلى كتابة ما يلي:

```
IF x > 65000 | y > 65000 THEN < خطأ >;
IF x ≠ 0 & y > (65000/x) THEN < خطأ >;
z = x × y.
```

وهذا هو السبيل الوحيد لتشخيص ما يحدث على وجه التحديد في

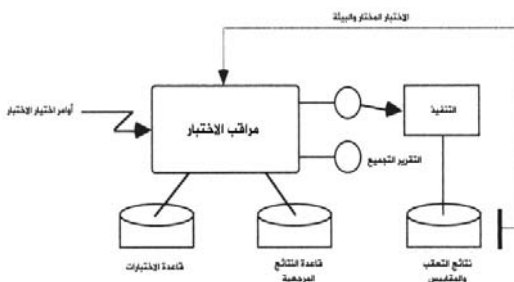
حال تجاوز قدرات البرنامج.

تمكننا إذن تغطية البرنامج^١ من التحقق من صحة النطاقات،

الأمر الذي يظهر مجدداً ثنائية الشفرة-البيانات. وفي حال ولد المترجم إلى لغة الآلة هذه الشفرة (وهو ما تتيحه لغات بالغة التميز مثل لغة أدا)، لا يمكن ضمان التغطية إلا إذا كان المترجم إلى لغة الآلة نفسه هو الذي ولد البيان، وهذا من القيود الأساسية للأدوات التي تعالج البرنامج المصدر.

٥- اختبارات الأداء: تتطوي هذه الاختبارات على التأكد من الأداء الإجمالي الجيد للبرنامج الحاسوبي في سياق بيئته. الجدير بالذكر أن هذه الاختبارات ذات أهمية خاصة بالنسبة للنظم الحاسوبية التي تطفئ عليها برامج من نوع ب.

٦- أدوات الاختبار: تلعب أدوات الاختبار دوراً بالغ الأهمية في سياق التكلفة الاقتصادية الإجمالية لمشروع صنع نظام حاسوبي (راجع. الفصل ٤، ٨ أنفاً). فبمجرد تجاوز المشاريع حجماً معيناً، يصبح مراقب الاختبارات مكوناً لا غنى عنه:



وتتطوي الوظائف التي يضمنها المراقب على مراجعة القواعد المختلفة

وتحديثها، وأرشفة واستعادة الاختبارات والنتائج، ومولد متغيرات البيئة، ومقارنة النتائج، ووظائف المحاكاة (الشاشات، الواجهات...) إلخ. وفي سياق الأدوات الأساسية، فإن اتساق ما تحدثه بيئة البرمجة (راجع الفصل ٤، الفقرة ٨، C) وبيئة التنفيذ عامل محوري مؤثر في الإنتاجية.

٧- استراتيجيات الاختبارات. - نظراً إلى الحالات التبادلية الكثيرة الناجمة، لا بد من وجود معايير تسمح باتخاذ القرار المناسب. لعل المعيار المنطقي هو المدة الوسطية بين حوادث خدمة النظام الحاسوبي (MTBSI)، أي مجموع المدة الوسطية قبل حدوث العطل + المدة الوسطية قبل الإصلاح (إتمام الإصلاح) ($MTTF + MTTR$)، وكذلك الجاهزية التي هي النسبة $MTTF/MTBSI$. ويتميز هذان البعدان بقياس وثوقية البرنامج كما يراها المستخدم بإضافة بعد سرعة إصلاح المستخدم للبرنامج. وهو المقياس الوحيد الذي يدل على شيء محدد، ألا وهو تكلفة الأعطال بالنسبة للمستخدم والمحرج الحريص على رضا عميله في آن واحد. أما الباقي فأدبيات نظرية. ينبغي أن تعظم استراتيجيات الاختبارات المدة الوسطية بين حوادث خدمة النظام الحاسوبي MTBSI فهو معيار بالغ الحزم بالنسبة لمحرج النظام البرمجي، إذ عليه وضع آلية قياس تسمح برصد تغيرات هذا المعيار (MTBSI).

الفصل السابع

إدارة مشاريع البرمجيات

أولاً: نموذج التكلفة التأسيسي (COCOMO)

هو نموذج* لتقدير التكاليف طوره العالم بي. بوهيم على أساس قاعدة من البيانات الإحصائية التي جمعت من عدد كبير جداً من المشاريع عامي ١٩٨٠-١٩٨١. ويعرض كتابه الكلاسيكي علم اقتصاد هندسة البرمجيات [١٧، ١] وصفاً للنموذج الكامل.

ويأتي هذا النموذج في ثلاثة مستويات سنعرض فيما يلي أسسها.

١- معادلات النموذج: لاحظ بي. بوهيم، كغيره من المؤلفين،

العلاقات الكمية التي تربط بعض الأبعاد التي تميز مشاريع البرمجيات
مثل:

- عدد التعليمات التي سلمت، بالألوف (ألف سطر من التعليمات المسلمة)

- إجمالي المجهود (أفراد-شهور)

- مدة التطوير

وحسب نوع البرامج ث، ب وم. يقترح النموذج العلاقات التالية:

ث___برمج: أفراد-شهور = ٢،٤ (ألف سطر من التعليمات المسلمة)^{١،٠٥}

مدة التطوير = ٢،٥ (أفراد-شهور)^{٠،٣٨}

ب___برمج: أفراد-شهور = ٣،٠ (ألف سطر من التعليمات المسلمة)^{١،١٢}

مدة التطوير = ٢،٥ (أفراد-شهور)^{٠،٣٥}

م___برمج: أفراد-شهور = ٣،٦ (ألف سطر من التعليمات المسلمة)^{١،٢٠}

* Cocomo :Constructive Cost Model

مدة التطوير = ٢,٥ (أفراد-شهور) ٠٣٢.

من ثم يتم تعديل هذه العلاقات وفق حجم البرامج.

٢- أسلوب التحصيل مرحلةً ونشاطاً: من ثم يعرض النموذج عدداً من مصفوفات تحصيل الجهد لكل مرحلة ونشاط في دورة الحياة. أما أوجه النشاط التي تؤخذ بالاعتبار فهي: تحليل المتطلبات، والتصميم، والبرمجة، وتخطيط الاختبارات، والتحقق والمصادقة، وإدارة المشروع، وإدارة التكوين وإدارة الجودة والتوثيق. وانطلاقاً من هذه المصفوفات، يمكننا إعادة تحصيل وتعديل هذه الأرقام وفق المنهج المتبع لمشروع خاص.

٣- الضبط النهائي للنموذج: يمكن الاكتفاء بالقيم التي يوردها النموذج الأساس إن كنا لا نطلب سوى القيم الأسية. إلا أن بوسعنا تعزيز تقديرنا للتكاليف وفق عدد من العوامل الثابتة ذات تأثير كبير على الإنتاجية، فالعوامل الـ ١٤ التي يراعيها النموذج هي: الخبرة بلغة البرمجة، القيود الزمنية، حجم البيانات، تفاعل مركز العمليات الحسابية، الخبرة ببيئة التشغيل، الخبرة بنموذج البرمجة، أدوات التطوير، تنفيذ البرمجة البنوية، القيود الخاصة باحتقان الذاكرة، الخبرة بالتطبيق، القيود الخاصة بمدة الاستجابة، القيود الخاصة بسلامة التشغيل، مدى تعقيد التطبيق، وكفاءات فريق العمل والأفراد. في هذه الدراسة، من المجدي مقارنة كيفية ترجيح النموذج لهذه العوامل:

البيان: ٧,٣

كفاءة الفريق: ١٨، ٤

أدوات التطوير: ٦٩، ٢

يخلص هذا النموذج إذن إلى أنه ينبغي أولاً أن يكون ثمة مهندسون جيدون، ومن ثم فريق تطوير ناجح وأخيراً بعض الأدوات التي حُسن اختيارها.

ثانياً : تنظيم عملية التطوير

التنظيم الإداري وفق المنتج/ وفق المشروع: يحقق تنظيم عملية التطوير وظيفتين أساسيتين:

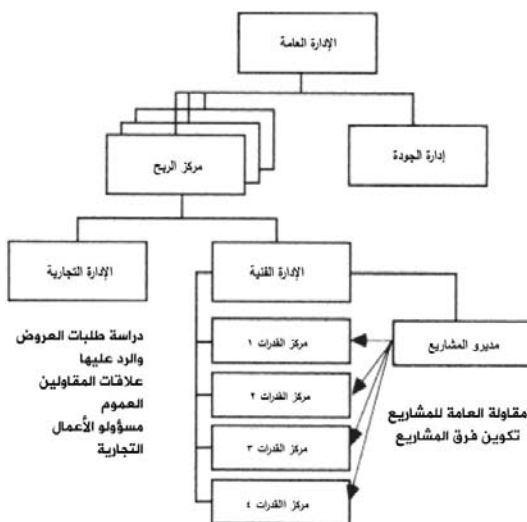
- وظيفة خاصة بتواصل جميع الأفراد المشاركين بشكل ما في سير المشروع (بضعة مئات عندما يكون المشروع كبير الحجم). ومن المفترض أن تكلف هذه المهمة الجهات المناسبة بمعالجة مختلف التحديات وأن تفسر القرارات المتخذة.
- وظيفة صنع القرار حينما توجد عدة حلول مناسبة لتحديد ما، وحينما ينبغي الاختيار بين تحقيق الربح المالي على المدى القريب والمتوسط والبعيد. ومن الشروط المسبقة لهذه الوظيفة، التنبه أولاً لوجود قرارات تمس المستقبل وإن بدت المشكلة المطروحة بادئ الأمر غير ذات أهمية.

هذا وينبغي تكيف أسلوب التنظيم مع طبيعة التحديات المتوقعة للمشروع، دون التردد قط في تغييرها مع تطور التحديات. ويمكن ملاحظة مواطن اختلاف تنظيمي بين الحزم البرمجية وبين برمجيات التسليم الجاهز.

يأتي التنظيم الإداري الخاص بإصدار حزمة البرمجيات على الشكل التالي:



لا بد أن تهتم الشركة المتخصصة بحزم البرمجيات بالاستمرارية،
وأيلاً تفضل قط تحقيق الربحية المباشرة على حساب مدة حياة حزمة
البرمجيات (إلا في حالات المراحل النهائية).
أما الشركة المتخصصة بإصدار برمجيات التسليم الجاهز (مفتاح
باليد)، فتأتي بالشكل التالي:



ينبغي للشركة المتخصصة بمشاريع التسليم الجاهز أن تحقق أعلى
مستوى من الربح في كل مشروع تجاري؛ فالتحكم بالتكاليف مهمة ذات
أولوية هنا. ويتم تطوير القدرات انطلاقاً من سعر تكلفة بعض الأعمال
التي تراها الإدارة العامة استراتيجية، إذ تكون القدرات على صعيد تقني
(قاعدة البيانات، الشبكات، إلخ..) وعلى صعيد قطاعي (مصرف،

مراقبة طيران، زمن حقيقي، إلخ..).

في حال المشاريع الضخمة التي تقتضي الاستعانة بعدة فرق عمل، من الأهمية بمكان تبني أسلوب تنظيمي يسهل تناول التحديات وإناطة مستويات القرار المناسبة بها. فهو فن بالغ الصعوبة يقتضي مرونة في التفكير وملكات عقلية عظيمة، وقدرة متميزة على التواصل مع الآخرين، وقدرة حقيقية على صنع القرار وتطبيقه. ولا يوجد ما هو أسوأ من عدم القرار، إذ لا بد عاجلاً أم آجلاً من اتخاذ القرار. وإن لم يكن أسلوب التنظيم ملائماً، تم اتخاذ القرارات في مستويات غير مناسبة، بل ومن قبل المبرمجين في أسوأ الأحوال، ما قد يلحق ضرراً فادحاً بالجانب الاقتصادي الإجمالي للمشروع؛ وهذا أحد الأسباب الأولى لفشل المشاريع الضخمة. إن حيابة مديري المشاريع المتمرسين شرط لا غنى عنه لنجاح أي عملية تطوير كبيرة.

ولجميع هذه المسائل بالغة الأهمية، نحيل القارئ إلى مؤلفينا :
إنتاجية المبرمجين (Productivité des programmeurs)
وتكاليف ومدة المشاريع المعلوماتية (Coûts et durée des projets informatiques) [١٧، ٢١].

الفصل الثامن

مستقبل هندسة البرمجيات

يمكن تمييز ثلاث فترات في تطور هندسة البرمجيات. فقد شهدت الفترة الأولى ظهور التقنيات والأساليب التي أتاحت برمجة أوائل التطبيقات المعلوماتية بشكل فاعل منذ أواخر الخمسينات. وقد سادت خلال هذه الفترة التي استمرت ٢٠ عاماً، لغات البرمجة ونظم إدارة قواعد البيانات، التي تشكل عماد هندسة البرمجيات. وكانت نهجاً مثل البرمجة البنيوية الاجتزائية، ونموذج علاقات الكيانات، من الثمار المحققة في هذه الفترة.

وبعيداً عن النجاح منقطع النظير الذي حققه النموذج العلائقي للبيانات (وحزم البرمجيات العديدة التي تدعمه)، فإن النتائج المحققة في مجال لغة البرمجة مغايرة تماماً. فمن المفارقات رجوع الثنائي سي/سي++ بعد سنوات عديدة من البحث والدراسة، وهما اللغتان التي أقل ما يمكن قوله إزاءهما أنها ليست أفضل ما أنجزناه. ولم تجد الملايين التي أنفقتها وزارة الدفاع الأمريكي نفعاً في دعم لغة أدا في هذا الصدد.

من ثم انطلقت مرحلة ثانية في مستقبل الثمانينات بظهور محطات العمل الأولى، إذ شاهدنا انتشار عروض أدوات بالغة التباين، التيسر خلالها صورة هندسة البرمجيات شيئاً ما. وقد رسخت حينئذ فكرة إمكانية «حل» أزمة هندسة البرمجيات بالأدوات وذلك من خلال:

- بتجاهل نتائج الثورة التي أحدثتها الشبكات وتقنية المعلومات الصغيرة*؛

- وتناسي حقيقة أن هذه الأدوات ستكون بين أيدي مستخدمي حقيقيين عاملين في مشاريع حقيقية لكل منها تحدياته الخاصة به. وأتت خيبة الأمل بحجم الأهداف البعيدة عن الواقع التي تصور

* الصغيرة = micro informatique

البعض أنها قريبة المنال.

بيد أن ثمة أدوات جوهريّة ابتُكرت خلال هذه الفترة، مثل إدارة التكوين، وأدوات هندسة البرمجيات بمساعدة الحاسوب لتطوير نظم المعلومات.

ثم كانت فترة ثالثة مع حقبة التسعينات، فترة تتسم بالعودة إلى الواقعية، الأمر الذي يتجسد في اتجاهين:

- عملية التطوير التي يحركها الفرد المنظم ضمن فريق عمل، هي الإطار المرجعي الذي لا غنى عنه الذي تتفعل فيه الأدوات والوسائل (وليس العكس كما ظن البعض بعيداً عن الحكمة)، وبات التكامل الآن يقوم بدور محوري تتبلور حوله سياسات الاقتناء وإعادة الاستخدام. وتتكامل عملية التطوير مع هندسة البرمجيات بمساعدة الحاسوب ؛
- يشكل كل من بنيان النظام البرمجي وسمة الوثوقية اللازمة له، التحدي الأول للسنوات المقبلة. وستقاس الأدوات والوسائل بهذا الواقع التقني المريع، التي لا فائدة من محاولة إخفائها لأن مسألة التقيد قائمة في جميع الأحوال.

لا يوجد سوى خيار واحد: فإما أن تسهل الأدوات والوسائل بشكل ملموس عمل مهندس البرمجيات، وتحكم الإدارة بالمشاريع، فتستمر، وإما أن تتلاشى بكل بساطة مع كل من يروج لها. وينبغي النظر إلى خيبة الأمل العظيمة التي أحدثها الذكاء الاصطناعي، والتي كان يمكن التنبؤ بها، كتذكارة مفيدة بحقيقة عالم لا سهولة فيه.

لعل التحدي الحقيقي الذي يواجه هندسة البرمجيات، هو العمل على إبقاء النظم المعلوماتية التي باتت تحيط بنا من كل صوب، والتي بتنا نتفاعل معها على نحو متزايد، في خدمتنا وطوع منطقتنا، فترتقي على وتيرتنا: ولا زال أمامنا الكثير الذي ينبغي إنجازه لتحقيق ذلك، وبخاصة في سياق التكامل والمصادقة والتحقق والاختبار، وهي الجوانب التي لا زالت السمات الأساسية للجودة.

تقنية المعلومات تعتمد أكثر من أي تقنية أخرى على مستوى إعداد مهندسيها، فكما كان الحال بالنسبة للمهندسين الالكترونيين في الخمسينات، الذين كان عليهم تعزيز معرفتهم الأساسية بفصول جديدة في مجال التحليل الرياضي، سيكون على المهندسين البرمجيين تطوير قدرتهم على تصور المفاهيم والتفكير التجريدي، والنمذجة، وتشديد نظم من الرموز، الأمر الذي يجعل تعليمًا معمقًا لعلم المنطق أمراً لا غنى عنه. كما ينبغي أن يحتل كل من تقصي أسباب الأخطاء، وتجنبها، الصدارة في هندسة البرمجيات. فإن كان لا مفر من وجود الأخطاء، فإن إنتهاء كل خطأ بخلل و/أو عطل أمر مهلك. كما أن دراسة مكثفة لمعايير الفعالية وغيرها من آليات سلامة الأداء شرط ضروري لموثوقية نظم المستقبل. كما أن كفاية هذا الشرط أمر لا يتوقف إلا علينا: فهذا هدف هام في هندسة البرمجيات في الأعوام القادمة.

المراجع

- [1] B. Boehm, Software engineering economics, Prentice Hall, 1981.
- [1 . 1] B. Boehm et al., Software cost estimation with COCOMO II, Prentice Hall, 2000.
- [2] M. M. Lehman, L. A. Belady, Program evolution, processes of software change, Academic Press, 1985.
- [3] D. Knuth, Art of computer programming, Addison-Wesley, 1968.
- [4] IEEE, Software engineering standards collection, et normes ISO 12207, 9126, 15504.
- [5] J. Musa, A. Ianino, K. Okumoto, Software reliability : measurement, prediction, application, McGraw-Hill, 1987.
- [6] W. Humphrey, Managing the software process, Addison-Wesley, 1989.
- [6 . 1] W. Humphrey, A discipline for software engineering, Addison-Wesley, 1995.
- [7] M. Shooman, Software engineering, design, reliability and management, McGraw-Hill, 1983.
- [8] I. Sommerville, Software engineering, Addison-Wesley, 1989.
- [9] R. Pressman, Software engineering : A practitioner's approach,

McGraw-Hill, 1992.

[10] B. Kernighan, P. Plauger, Element of programming style, McGraw-Hill, 1974.

[11] C. Jones, Programming productivity, McGraw-Hill, 1986.

[11 . 1] C. Jones, Estimating software costs, Mac Graw-Hill, 1998.

[12] J. Kowal, Analysing systems, Prentice Hall, 1986.

[13] P. Denning, J. Dennis, J. Qualitz, Machines, languages and computation,

Prentice Hall, 1978.

[13 . 1] Z. Mammeri, SDL, Hermès-Lavoisier, 2000. SDL est un langage normalisé par l'UIT/CCITT depuis 1993.

[14] K. Nielsen, K. Shumate, Designing large real time systems with Ada, McGraw-Hill, 1988.

[15] D. Tchiritzis, F. Lochovsky, Data models, Academic Press, 1982.

[15 . 1] T. Teorey, Databases modelling and design, Morgan Kaufmann, 1999.

[15 . 2] J. Gray, A. Reuter, Transaction processing : Concepts and techniques,

Morgan Kaufmann, 1993.

[16] B. Beizer, Software testing techniques, Van Norstrand Reinhold, 1989.

[16 . 1] R. Binder, Testing object-oriented systems, Addison-Wesley,

2000.

[17] J. Printz, Productivité des programmeurs et Coûts et durée des projets

informatiques, 2 vol. chez Hermès-Lavoisier, 2001.

[18] J. Printz, Puissance et limites des systèmes informatisés, Hermès-Lavoisier, 1998.

[19] B. Meyer, Conception et programmation orientées objet, Eyrolles, 2000.

[20] OMG, Unified Modeling Language UML, Version 1.3, 1999, et s.

[21] J. Printz, Écosystème des projets informatiques, Hermès-Lavoisier, 2005.

قائمة المحتويات

٥	مقدمة
٧	الفصل الأول: بعض المعطيات الاقتصادية
١٣	الفصل الثاني: تصنيف البرمجيات
٢١	الفصل الثالث: القضية الأساسية في هندسة البرمجيات
٢٧	الفصل الرابع: نموذج التطوير ودورة الحياة
٦١	الفصل الخامس: حركية دورة التطوير وديناميكيته وتنظيمها
١٠١	الفصل السادس: البرمجة والاختبارات
١٣٩	الفصل السابع: إدارة مشاريع البرمجيات
١٤٥	الفصل الثامن: مستقبل هندسة البرمجيات
١٤٩	المراجع:

عن الكتاب:

هندسة البرمجيات علم يهدف في نهاية المطاف إلى صنع أنظمة معلوماتية غالباً ما تتسم بالتعقيد. ويُعين هذا التخصص المعرفي الشمولي القواعد والحدود -التي هي بمثابة شروط ضرورية- لسير عملية صنع هذه الأنظمة بشكل سليم؛ لذلك لا بد أن تنبثق هذه القواعد والحدود من الواقع العملي لتطوير البرمجيات، وأن تسعى دوماً إلى تحقيق عوامل فاعلية عملية التطوير ودلالة الأدوات التي تقترحها. ويمكن هذا المؤلف القارئ -إذ يرسم مشهداً كاملاً للبيانات الاقتصادية والفنية لتطوير البرمجيات- من تعزيز إدراكه للتحدي الحقيقي الذي تواجهه هندسة البرمجيات؛ وهو ضمان بقاء كافة الأنظمة المعلوماتية التي تحيط بنا من كل صوب قيد خدمتنا.

المؤلف:

جاك برينتز

جاك برينتز هو أستاذ في المعهد الوطني للفنون والحرف.

المترجم:

أ.زينا مغربل

حاصلة على بكالوريوس في إدارة أنظمة المعلومات من جامعة مرييلاند الأمريكية. تعمل في تعريب وترجمة العديد من المؤلفات مثل الكتب والمجلات من اللغات الإنجليزية والفرنسية، وبخاصة في مجالات العلوم والتقنية.



LE GÉNIE LOGICIEL

Jacques Printz

pufl



مدينة الملك عبدالعزيز
للعلوم والتقنية KACST

تعمل مدينة الملك عبد العزيز للعلوم والتقنية على توفير المعرفة للقارئ العربي. فقامت في هذا الإطار بنشر سلسلة من الكتب والمجلات العلمية وأتاحتها للقراء دون مقابل بصيغتها الرقمية والورقية. فجميع إصدارات المدينة متاحة على موقعها الإلكتروني ليتمكن المتصفح من تحميلها أو قراءتها على الإنترنت.

www.kacst.edu.sa
publications.kacst.edu.sa
awareness@kacst.edu.sa

الموقع الإلكتروني:
إصدارات المدينة:
البريد الإلكتروني:

هاتف: ٠١١ ٤٨٨٣٤٤٤ - ٠١١ ٤٨٨٣٥٥٥
فاكس: ٠١١ ٤٨٨٣٧٥٦
ص.ب. ٦٠٨٦ الرياض ١١٤٤٢
المملكة العربية السعودية
مدينة الملك عبد العزيز للعلوم والتقنية

