الفصل الخامس
تصميم البرامج

يتميز المبرمجون بالصبر والهدوء اللازمين للبرمجة فالتصميم السليم ضرورى للبرنامج الناجح ، وقد تم منح اسم (تصميم النظام وتحليله System Analysis and Design) على : عملية تحليل المشكلة ، ثم تصميم البرنامج لهذا التحليل .

تصميم البرنامج مهم للغاية ، وهناك ثلاث خطوات أساسية مطلوبة عند كتابة برنامج ، يعد تصميم تعريف المخرجات Output Definition هاما ، ويكون تصميم (أعلى أسفل) Top - Down أفضل من تصميم (أسفل أعلى) Up–Bottom .

تستخدم خريطة أو هيكل التدفق Flowchart بالعلامات الموجودة داخلها لتصميم البرنامج كما قد يستخدم الكود النصى الذى قد يتميز عن خريطة التدفق .

فهم تصميم البرنامج

عند بناء منزل يجب تصميمه أولا ، ولا بد أيضا من تصميم البرنامج قبل كتابته ، ويجب قبل كل شىء معرفة ما يريده مشترى البرنامج ، لذلك يجب إجراء مقابلة مع المصمم لمعرفة ما يدور بذهن المشترى عن شكل البرنامج ، ويقوم المصمم بإرشاد المشترى عن مدى إمكان تحقيق مطالبه ، وأثناء المرحلة المبدئية يعتبر السعر هو العامل الذى يربط دائما بين المشترى والمصمم من أجل الوصول إلى اتفاق مشترك .

بعد انتهاء المصمم من تخطيط البرنامج يجب أن يخطط المبرمج الموارد اللازمة لبناء هذا البرنامج ، وبمجرد انتهاء التصميم يتم شراء الموارد ، ويبدأ العمل الحقيقى ، وكلما كانت الخطط المبدئية واضحة كلما قلت المشكلات المحتمل ظهورها فى المستقبل .

يجب دائما تذكر هذا قبل كتابة برنامج ، فلا يجب الاتجاه إلى لوحة المفاتيح والبدء فى كتابة قواعد وأوامر البرنامج قبل تصميمه ، فكلما بذلت مجهودا كبيرا فى التصميم كلما انتهت من البرنامج سريعا .

جعلت تكنولوجيا الكمبيوتر تعديل البرنامج سهلا إذ يمكن الإضافة إلى البرنامج

وتعديله بسهولة ، ومع ذلك فإضافة شئ إلى برنامج لا يعد فى سهولة تصميم البرنامج بشكل سليم منذ البداية .

تعتبر عملية صيانة البرمجة التى تأخذ مكانها بعد كتابة البرامج واختبارها وتوزيعها من أهم أساسيات عملية البرمجة ، وتتم صيانة البرامج باستمرار لتعكس احتياجات المستخدم ، ففى بعض الأوقات إذا لم يتم تصميم البرنامج بشكل سليم قبل كتابته يرفض المستخدم هذا البرنامج إلا بعد أن يؤدى ما يطلبه منه .

مبدئيا هناك ثلاث خطوات يجب أداؤها عند كتابة برنامج :

١- تعريف المخرجات .

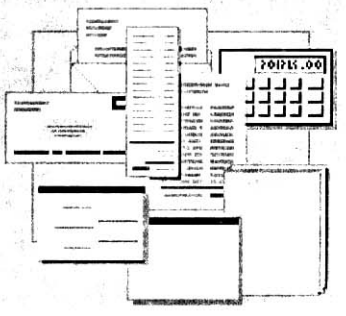
٢- إنشاء المنطق الخاص بالحصول على تلك المخرجات .

٣- كتابة البرنامج .

كتابة البرنامج هى الخطوة الأخيرة ، وهذا لا يبدو غريبا ، وإذا تم التفكير فى التصميم بعناية فتصبح كتابة البرنامج عملية تلقائية وسهلة .

الخطوة الأولى : تعريف المخرجات

يجب قبل بداية العمل فى برنامج معرفة ما سوف ينتجه هذا البرنامج (المخرجات Outputs) ، وتعد المخرجات أخر شئ ينتج عن البرنامج لكنها تعد الشىء الأول فى مرحلة التصميم ، فيجب على المبرمج معرفة ما ستكون عليه المخرجات قبل كتابة البرنامج .



تعد مخرجات برنامج أكثر من مجرد معلومات مطبوعة فقط (على الشاشة أو على أى وسيط إخراج) ، فأى شئ ينتجه البرنامج ، أو يراه المستخدم ، يعتبر مخرجات يجب تعريفها ، كما يجب أيضا أن يكون المبرمج على دراية بكل شاشة معروضة

أمام المستخدم فى البرنامج ، وبكل ورقة مطبوعة ، وبشكل كل تقرير من التقارير التى ينتجها البرنامج ، فذلك يعطى نظرة فاحصة إلى عناصر البيانات التى يقوم الجهاز بتتبعها ، أو حسابها ، أو إنتاجها ، ويساعد على تعريف المخرجات أيضا ، وأيضا يساعد على تجميع البيانات التى يحتاجها المبرمج للحصول على المخرجات .

يحتوى تعريف المخرجات على كثير من التفاصيل ، ويجب تحديد جميع تفاصيل المشكلة قبل تحديد المخرجات التى يريدها المبرمج ، ويعد التصميم من أعلى إلى أسفل من أفضل طرق تحديد تفاصيل مشكلة معينة .

تنتج بعض البرامج كميات ضخمة من المخرجات لذلك لا يجب إهمال هذه الخطوة أو عدم توفير الوقت المخصص الكافى لها لأنه مع ازدياد حجم المخرجات يصبح لازما تعريفها ، ويتسم تعريف المخرجات بالسهولة إلا أنه فى بعض الأحوال يستهلك الوقت ويكون مملا ، وقد تستغرق عملية تعريف المخرجات وقتا يماثل خطوة كتابة البرنامج (وتستهلك الخطوة الثانية وقتا أكثر من كل الخطوات) ، ومع ذلك فسوف تستهلك وقتا أكثر عند إهمال تعريف المخرجات فى البداية .

تصميم البرنامج من أعلى إلى أسفل Top-Down

يعتبر تصميم البرنامج من أعلى إلى أسفل Top Down من أهم التصميمات المتاحة فهو يساعد على الحصول على التفاصيل اللازمة لإتمام عملية البرمجة ، وهو (التصميم من أعلى لأسفل) عملية تقسيم للمشكلة بالكامل إلى تفاصيل صغيرة دون حصر كل هذه التفاصيل ، ويتجه المبرمجون إلى عدم استخدام هذا النوع من التصميم لاستهلاكه الوقت ، ويفضلون الاتجاه العكسى الآخر (التصميم من أسفل إلى أعلى Bottom-Up) ، وفى هذه الحالة يضع على نفسه عائقا كبيرا لتذكر جميع التفاصيل اللازمة فهذا النوع من التصميم يحصر جميع التفاصيل .

أحد مفاتيح التصميم من أعلى إلى أسفل Top-Down هو أنه يجبر على ترك التفاصيل إلى حين توقيت الحاجة إليها ، فهو يحث على النظر إلى المشكلة ككل فى تركيز كامل ، أما فى حالة التصميم من أسفل إلى أعلى Bottom - Up فتحصل على التفاصيل بسرعة ، وتفقد الأهداف الأساسية للبرامج .

يمكن تعلم المزيد عن التصميم من أعلى إلى أسفل Top-Down بربطه مع مشكلة

واقعية قبل التعرض لمشكلة كمبيوتر ، فهو لا يقتصر على مشكلات البرمجة إذ يمكن تطبيقه على أى جزء شيء تريد تخطيطه بدقة وتفصيل .

خطوات التصميم من أعلى لأسفل Top-Down

تتمثل الثلاث خطوات الرئيسية للتصميم من أعلى لأسفل Top-Down فى :

١- تحديد الهدف العام (برنامج المرتبات) .

٢- تقسيم هذا الهدف إلى أجزاء تفصيلية (التفاصيل الكثيرة تدفع إلى ترك أشياء) ، (الموظف وتكوين المرتب ، والخصومات) .

٣- ترك التفاصيل جانبا قدر الاستطاعة ، ثم تكرار الخطوتين الأولى والثانية على باقى المكونات (الموظف : الاسم والعنوان والوظيفة والدرجة وتاريخ التعيين وتاريخ الترقى ، .. ، تكوين المرتب : الأساسى والعلاوات بأنواعها ، والحوافز بأنواعها ، والخصومات والضرائب والاستقطاعات ، ..) حتى الوصول إلى حد لا يمكن معه تقسيم المشكلة أكثر من ذلك .

تتمثل الخطوة الثانية فى التصميم من أعلى لأسفل Top-Down فى أخذ مكونات جديدة ، وعمل نفس الشيء لكل منها ، وينتج تصميم Top-Down نتيجة على شكل مثلث يظهر الجزء الأول منها مثال يعرض التخطيط للزواج فهناك الكثير من التفاصيل لم توضع بعد لكن هذه هى فكرة ترك التفاصيل جانبا أكبر وقت ممكن ، وستأتى التفاصيل أثناء تقسيم المهام إلى أجزاء أكثر .

بالتأكيد فالهدف العام الخاص يشمل الفكرة العامة التى تتعلق بموضوع البدء ، وستأتى التفاصيل فى النهاية ، وعلى الفور ستجد أن حجم الورقة قد ازداد ، ويمكن استخدام مزيد من الأوراق وترقيم الصفحات ووضع رقم الصفحات فى المربع الذى يعلو الصفحة فى تصميم أعلى لأسفل Top-Down .

فى الواقع فإن تتبع الصفحات ليس مهما فالصفحة الأولى هى الصفحة الأساسية التى يجب أن تتأكد أنها تعمل فى اتجاه الهدف الذى تريده ، وتستمر فى تقسيم الصفحات المفصلة إلى أن تنتهى عملية التقسيم ، وستجد فى النهاية أن كل تفصيل قد تم أخذه فى الاعتبار .

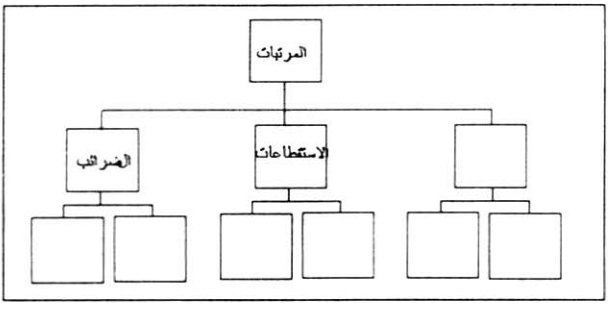
لا تقلق من ترتيب وقت التفاصيل فالتصميم من أعلى لأسفل Top-Down يستخدم فى

إنتاج التفاصيل التى تحتاجها (فى النهاية) ، وليس وضع هذه التفاصيل فى أى ترتيب ، يجب أن تعرف من أين تبدأ ، وما هو المطلوب بالضبط قبل أن تعرف علاقة هذه التفاصيل ببعضها البعض ، وأى منها يأتى أولا .

فى النهاية ستحصل على عدة صفحات من التفاصيل التى لا يمكن تقسيمها أكثر من ذلك .

بالانتقال إلى مشكلة تصميم برنامج الكمبيوتر نفترض مهمة كتابة برنامج مرتبات شركة ، فما هى متطلبات البرنامج؟ يمكن البدء فى سرد تفاصيل برنامج المرتبات على النحو التالى :

* طباعة شيكات المرتبات . * حساب الاستقطاعات . * حساب الضرائب .



إن التفاصيل تأتى مبكرا فالمكان الأمثل للبدء هو القمة ، ويتمثل الهدف العام من برنامج المرتبات فى إجراء المرتبات .

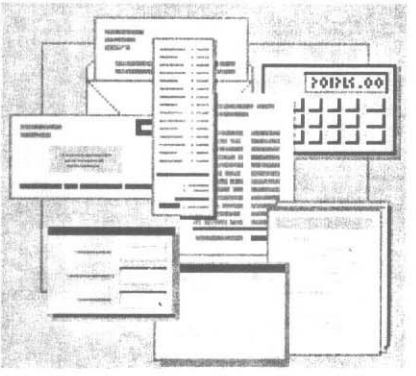
قد تكون هذه هى الصفحة الأولى لتصميم Top-down للمرتبات ، ويجب أن يشتمل أى برنامج على نظام إدخال ، وحذف ، وتغيير معلومات الموظفين مثل العنوان ، المدينة ، المحافظة ، الإعفاءات ، والبيانات الأخرى .

ما هى التفاصيل الأخرى التى تريدها عن الموظفين ؟ قف عند هذا الحد ولا تسأل ، فالتصميم لن يكون جاهزا لكل هذه التفاصيل .

هناك طريق طويل قبل الانتهاء من التصميم من أعلى لأسفل Top-down للمرتبات ويعتبر شكل البداية هو الخطوة الأولى التى يجب الاستمرار بعدها فى تقسيم كل جزء حتى تظهر التفاصيل فى النهاية ، ولا يمكن تحديد ما ينتجه البرنامج إلا فى حالة الحصول على جميع التفاصيل .

أدوات تعريف المخرجات

إن التفاصيل الناتجة عن التصميم من أعلى لأسفل Top-down ليست كلها مخرجات فالعديد من هذه التفاصيل تمثل إجراءات يجب أداؤها بواسطة البرنامج النهائى فعلى سبيل المثال : يعتبر حساب صافى الأجر أحد تفاصيل برنامج المرتبات ، وهو يؤدى إلى عدم ظهور مخرجات ، وفى مكان آخر فى التصميم من أعلى لأسفل Top-down يظهر تفصيل "طباعة صافى الأجر على شيك الدفع" ، ويعد هذا التفصيل مخرجات يجب التأكد من مكوناتها وأنها تبدو كما يريدها المستخدم أن تظهر فى البرنامج النهائى ، فوظيفة المبرمج هى الاتفاق مع المستخدم ، وتعريف كل عنصر من مخرجات البرنامج ، وهذا ما يسمى بتعريف المخرجات .



يمكن إنشاء نموذج لمخرجات البرنامج باستخدام أدوات البرمجة المتوفرة مثل فيجوال بيزك Visual Basic ، ويمكن للمبرمج إنشاء شاشة فى بضع دقائق ثم يعرضها على المستخدم تلقائيا بعد ذلك للتأكد من أن المخرجات تشتمل على البنود المطلوبة .

من مميزات نماذج الشاشات التى تقدم للمستخدم أثناء مرحلة التصميم باستخدام الأدوات المتقدمة المتاحة هى : أنه بمجرد أن يتم الاتفاق بين المبرمج وبين المستخدم على تصميم شاشة معينة يمكن بسهولة إحضار هذه الشاشة المصممة وإدخالها ضمن البرنامج النهائى ، بمعنى أن نموذج الشاشة يمكن أن يصبح شاشة حقيقية فى المنتج النهائى ، وإذا لم تستخدم أدوات وفضلت استخدام أدوات تصميم معتمدة على الورق لمخرجات البرنامج سيكون واجبا إدخال تصميم الورق داخل الجهاز عند كتابة البرنامج ، وستتمكن من توفير الوقت إذا أصبح التصميم نفسه تلقائيا وجزءا من البرنامج .

يعد النموذج التجريبى ورقة عمل خالية لا يمكنها عمل شئ لكنها تصور تفاعل المستخدم معها وذلك حتى يتم ربط أجزائها بشفرة البرنامج ، وتبدأ وظيفة المبرمج بعد الحصول على موافقة المستخدم على الشاشات التجريبية ، وتعتبر الشاشات هى المكان الأول للبدء لأنه يجب معرفة ما يريده المستخدم قبل الاستمرار فى العمل .

عند استخدام أداة برمجة متقدمة مثل فيجوال بيزك Visual Basic التى يستخدمها معظم المبرمجين فإنها تبين كيفية تمثيل الشاشات ، فعلم البرمجة ليس ضروريا لتصميم الشاشات ، وبهذه الطريقة سيكون للمستخدمين القدرة على إظهار ما يريدون بالضبط ، كما أن الشاشات التجريبية تتميز بالتفاعل بمعنى أن المستخدم سيتمكن من نقر الأزرار وإدخال القيم فى الحقول على الرغم من عدم حدوث شئ نتيجة لذلك ، والغرض هنا يتمثل فى أن المستخدم يقوم بتجربة الشاشات لفترة معينة ليتأكد من ملاءمتها له ووضع المتحكمات بها .

أحيانا لا يمكن استخدام أدوات للنماذج المتقدمة لأن جهاز المستخدم لا يملك الطاقة الكافية أو المساحة الكافية على القرص لتشغيل متطلبات البرمجة أو قد لا يملك تصريح تثبيت متطلبات البرمجة على جهازه ، وعند كتابة برنامج لمستخدم لا يعرف الكثير من المعلومات عن الكمبيوتر يجب عرض الشاشات عليه كما يجب الحصول على توقيعه على شكل هذه الشاشات حتى يضمن المبرمج استجابة المستخدم لتصميم البرنامج ، ويضمن عدم التراجع .

هناك عدة أدوات معتمدة على الورق تساعد فى تعريف المخرجات ، ويعتبر ورق الرسم البيانى مفيدا فى تصوير تفاصيل كل شاشة وتقرير ، إذ يبين ورق الرسم البيانى عدد سطور الشاشة ، والمساحات التى تفصل العناصر على الشاشة ، ويعطى الورق البيانى للمستخدمين فكرة عن الشاشات التى سوف يستخدمونها لكنها تعد أقل نموذجية مما تبدو عليه الشاشة الحقيقية ، لكن عمل رسم بيانى لعناصر المخرجات يساعد على معرفة مكانها النهائى فى البرنامج .

مع مرونة أنظمة التشغيل نجد المستخدمين الجدد للجهاز يألفون استخدام برامج معالجة النصوص للكتابة ، ويمكن استخدام الشاشات البدائية لمعالج النصوص لعرض الشاشات على المستخدم ، وبالطبع لا يمكن مقارنة نظام معالج النصوص بأدوات التصميم المتقدمة التى تحتوى على أدوات تصميم فعالة لكنه يعد أداة من أدوات العرض ، وبالطبع يمكن استخدام وسائل عرض أخرى من البرامج .

تأكد من أن تعريف المخرجات يشتمل على شاشات إدخال البيانات ، ويبدو هذا متناقضا لكن تتطلب شاشات الإدخال أن يقوم البرنامج بوضع أسئلة واستفسارات (أسئلة وعناوين البيانات المطلوبة) على الشاشة ، ويجب تخطيط المكان الذى ستذهب إليه عناصر شاشات الإدخال .

على سبيل المثال قد يبدو شكل شاشات الإدخال فى برنامج مخزن كتب أكثر من مجرد إدخال بيانات حيث يمكن إدخال بيانات كتب جديدة (بالإضافة إلى أن نفس الشاشة تنتج بيانات (مخرجات) الكتب المتواجدة) ، وبالرغم من أن المستخدم النهائى سوف يدخل عناصر بيانات الكتاب المطلوبة إلا أن تعريف شكل إدخال البيانات بالشاشة قبل كتابة البرنامج الذى ينتج الشاشة سيسرع من عملية انتهاء البرنامج .

الخلاصة أن شاشات المخرجات والتقارير المطبوعة وشاشات إدخال البيانات يجب أن يتم تعريفها ، وحتى يعرف المبرمج متطلبات البرامج يجب أن يعرف أيضا البيانات الواجب تواجدها فى الملفات ، علاوة على تنسيق الملفات ، وكلما تمرس المبرمج فى علم البرمجة كلما كان بإمكانه تعلم طرق لوضع ملفات الأقراص فى التنسيق المطلوب .

العمل مع المستخدمين

يجب التحدث مع المستخدمين وتعريف المخرجات بناء على احتياجاتهم ، وبالنسبة للمستخدمين عادة تكون فكرة العمل الذى سيؤديه البرنامج مبهمة ، لذلك يجب مساعدتهم فى معرفة عمل البرنامج وأيضا معرفة مخرجات البرنامج .

يجهل بعض المستخدمين قدرات الجهاز والبرامج فى العمل ، لذلك يجب إنشاء بعض مهارات الإرشاد للمستخدم أثناء تصميم البرنامج فيجب أن تكون قادرا على تحويل متطلبات البرنامج الخاصة بالمستخدم ككل إلى أجزاء وأشياء ذات معنى ، ويعد تعريف المخرجات من أفضل أدوات تعريف متطلبات البرنامج الخاصة بالمستخدم فيمكن عرض تصميم المخرجات على المستخدم ثم سؤاله أسئلة مثل : هل هذا هو ما تريده ؟ ، وهل يتضمن هذا التقرير كل شئ تحتاج إليه ؟ .

عند كتابة وبرامج لشركة ما فهى عادة تكون لها متطلبات خاصة صارمة عن محتويات كل تقرير مثال : وجود تاريخ ووقت طباعة التقرير فى كل صفحة مطبوعة من التقرير لذلك يجب تحقيق مقاييس مخرجات كل شركة لاستيفاء متطلباتها .

عند التعامل مع المستخدمين يجب إعطاؤهم أكبر عدد من النماذج التجريبية ليعملوا

بها ، فعند كتابة برنامج يعرض مخرجات وشاشات إدخال بيانات يستطيع المستخدم استخدامها بواسطة لوحة المفاتيح دون أن يقوم البرنامج بعمل أى شئ بالبيانات فعن طريق التعامل مع الشاشات فقط سيحصل المستخدم على صورة واضحة عما إذا كان البرنامج يشتمل على كل العناصر المطلوبة أم لا .

يجب عمل اتفاقية مكتوبة مع المستخدم تفيد بصحة تعريف المخرجات ، ويجب أن تتم هذه الخطوة المهمة قبل بداية العمل فى البرنامج ، وبهذه الطريقة نضمن عدم حدوث منازعات أو اعتراضات من المستخدم عند تقديم البرنامج النهائى .

ماذا ستفعل بكل التفاصيل الخاصة بتصميم Top down وتعريف المخرجات ؟ يجب تحويلها إلى أوامر مرتبة ترتيبا منطقيا وزمنيا حتى يستطيع البرنامج اتباعها للحصول على هذه التفاصيل ، وهنا يأتى دور خريطة التدفق والكود الشكلى .

الخطوة الثانية : توليد منطق البرنامج

بعد الاتفاق مع المستخدم على الأهداف ومخرجات البرنامج ، يكون الباقى متروكا للمبرمج ، فوظيفته الآن هى أخذ تعريف المخرجات ثم تحديد كيفية قيام الجهاز بإنتاج المخرجات ، وليس معنى قيامه بأخذ المشكلة العامة وتقسيمها إلى تعليمات تفصيلية أن يبدأ فى كتابة البرنامج ، بل على العكس فهو يكون الآن مستعد لتوليد المنطق الذى سينتج تلك المخرجات .

يلعب تعريف المخرجات دورا كبيرا فى شرح عمل البرنامج ، والآن يجب أن يقرر المبرمج كيفية إنجاز العمل ، وذلك عن طريق ترتيب التفاصيل حتى يمكنها أن تعمل فى شكل ترتيب مؤقت كما يجب أيضا أن يحدد القرارات التى يجب على البرنامج أداءها والإجراءات الناتجة عن كل هذه القرارات .

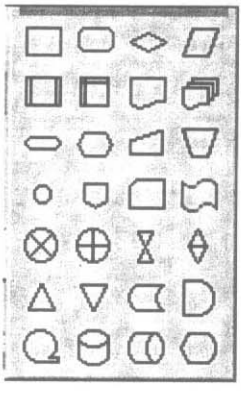
خرائط التدفق Flow Charts

تعد خرائط التدفق وأدوات المنطق المتعلقة بها إحدى النقاط الأساسية لمتخصص البرمجة ، لكن خرائط التدفق تدرس بقلة بالرغم من أن صعوبة البرامج تتطلب وجود طريقة لمعرفة مفهوم البرنامج أو منطقة قبل كتابته ، وتبدو خريطة التدفق قديمة لكن المبرمجين الذين لا يستخدمون أدوات مثل خرائط التدفق يكونون أقل إنتاجية مما هو الحال عند استخدامها .

بالنسبة لخريطة التدفق فإنها لا تشمل جميع تفاصيل البرنامج لكنها تمثل تدفق المفهوم

العام للبرنامج ، وتمنح مفهوم البرنامج النهائى ، وبعد اكتمال البرنامج النهائى تلعب خريطة التدفق دورا توثيق للبرنامج نفسه .

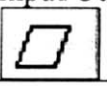
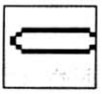
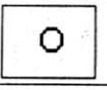
تشرح خريطة التدفق نموذج تقديم المدخلات والمخرجات لبرامج الجهاز ، وتتكون خريطة التدفق من رموز قياسية ، ويمكن شراء مسطرة بها رموز تدفق تسمى نماذج خريطة التدفق Flowchart Templates من المتاجر وذلك لتساعد على رسم خرائط تدفق أفضل من الرسم اليدوى ، وهناك برامج تساعد على إنشاء خرائط التدفق وطباعتها .



لا يمكن عمل خريطة تدفق لبرنامج إلا بعد تعلم البرمجة ، وعلى ذلك يجب تعلم خرائط التدفق (أو استخدام نوع آخر) قبل كتابة البرامج ، وتعتبر هذه الطريقة مفيدة بالنسبة للمبتدئين فى عالم البرمجة ، وعند كتابة برامج خاصة ستدرك أهمية الحاجة إلى خرائط التدفق .

هناك العديد من رموز خرائط التدفق لكن القليل منها يستخدم ، ويجب دراسة هذه الرموز بسبب انتشارها فى عالم البرمجة حيث يمكن استخدامها لرسم تفصيل أى حدث لا يوجد شئ يربط بين رموز خرائط التدفق ومربعات التصميم من أعلى لأسفل Top-Down .

عملية Process 	تشمل شرحا بالأشياء التى تمت يمكن استخدام رمز عملية Process عند معالجة بيانات مثل الحساب .
قرار Decision	يستخدم عندما يجب على البرنامج أخذ قرار معتمد على

	مخرجين مثل الطباعة على الشاشة أو على الطابعة .
مدخلات / مخرجات Input/Output 	تستخدم لأى مدخلات او مخرجات خاصة بالبرنامج مثل سؤال المستخدم سؤال معين أو طباعة تقرير .
وحدة طرفية Terminal 	إن رمز وحدة طرفية Terminal مع الكلمة ابدأ Begin أو البداية Start مكتوبة داخلها دائما تبدأ كل هيكل تدفقى كما أن رمز وحدة طرفية مع الكلمة النهاية END أو إنهاء Finish مكتوبة داخلها دائما تنهى كل خريطة تدفق .
تكملة على صفحة أخرى off-page 	يوضع توصيل off-page عند أسفل أى خريطة تدفق يكون موصلا لصفحة أخرى ، ويوضع رقم الصفحة التالية داخل موصل off-page ، ويوضع موصل off-page عند بداية كل صفحة تتضمن هيكلا تدفقيا خاصا بصفحة سابقة ، وأخيرا يوضع رقم الصفحة السابقة داخل موصل off-page الذى يبدأ صفحة جديدة .
توصيل التدفق Flow connector 	يستخدم فى حالة إدماج تدفق خاص بخريطة تدفق ، وستشاهد حرفا أبجديا مماثلا داخل توصيلة التدفق ، وتقوم توصيلة تدفق بملائمة تحديد نقطة إعادة الإدخال للفكر المتواجد .
سهم اتجاه التدفق Flow direction	تقوم هذه الأسهم بتوصيل كل رمز فى الهيكل التدفقى وتحدد اتجاه تدفق البرنامج .

قواعد خرائط التدفق

على الرغم من أن جميع المبرمجين يرسمون خرائط التدفق بطريقة مختلفة فهناك بعض القواعد المحددة التى يجب الاطلاع عليها ، وتتبع هذه القواعد على مستوى العالم لذلك يجب فهمها حتى تصبح خرائط التدفق التى تكتبها مقروءة بواسطة الآخرين .

القاعدة الأولى : استخدام رموز خرائط تدفق قياسية فاستخدام رموز متعارف عليها

يجعل الآخرين قادرين على فهم الهيكل التدفقى للخريطة ، ويستطيع المبرمج فهمها ومراجعتها بسهولة .

القاعدة الثانية : يجب أن يتدفق هيكل التدفق من أعلى الصفحة إلى أسفلها ، ومن اليسار إلى اليمين ، وإلا سوف تصبح غير منظمة ويصعب تتبعها ، وقد لا تحتاج بعض خرائط التدفق التحرك لليمين لأنها تشرح فكرا متتابعا للبرنامج لكن معظمها له تدفق تجاه أحد اتجاهين ، وهناك أوقات تقوم فيها خريطة التدفق بكسر هذه القاعدة بسبب تكرار المفهوم الذى يجعل هيكل التدفق يعود لمناطق فى أعلاها ويسارها ليكرر أقساما لكن فى النهاية يجب أن يستمر المفهوم فى الاتجاهات المطلوبة .

القاعدة الثالثة : تحتوى معظم رموز خريطة التدفق على نقطة إدخال وخروج واحدة ، ويحدد اتجاه التدفق للأسهم نقاط الإدخال والخروج ، ويعد رمز اتخاذ قرار Decision الرمز الوحيد الذى قد يكون له أكثر من نقطة خروج Exit Point ، وعادة يكون له نقطتان لأنه عند هذا المكان يحدث أحد أمرين ، ويتم تحديد التدفق التالى عن طريق نتيجة هذا القرار .

القاعدة الرابعة : يجب دائما على رمز Decision أن يسأل سؤالا بكلمات تشرح ما يحدث عند هذه النقطة من الهيكل التدفقى ، ويجب تسمية نقاط الخروج الخاصة برمز كل قرار بوضوح لأنها تعد نتيجة سؤال ، ويمكن تسمية المخرجات بنعم Yes أو y أو لا No أو N لمعرفة ما ترمز إليه المخرجات ، وهناك أوقات يجب على البرنامج الاختيار بين عدة قيم اعتمادا على البيانات التى يستقبلها .

القاعدة الخامسة : يجب صنع خريطة تدفق قبل كتابة برنامج ، ويجب أن يكون شرح التعليمات داخل الرموز بلغة واضحة (ليست جمل كمبيوتر أو لغة برمجة) ، وإلا لن يكون هناك داع لاستخدام هيكل خريطة التدفق ، ويقوم المبرمج فى النهاية بتحويل هيكل التدفق إلى جمل لغة برمجة بعد التأكد من قيام هيكل التدفق بأداء المنطق الذى يحتاجه .

الكود الشكلى أو الزائف Pseudo Code

تستهلك خريطة التدفق ورقا ووقتا كثيرين ، وهى محددة ولا تمنح المرونة التى

يحتاجها منطق البرمجة ويجب رسمها يدويا ، وعند انتهاء خريطة التدفق ، واكتشاف ترك رمز يجب إعادة رسم جزء كبير من هيكل التدفق .

على الرغم من قوة هيكل التدفق وسهولته إلا أن هناك بعض الشركات تفضل وسيلة أخرى لوصف المنطق تسمى الكود الشكلى Pseudocode ، وهو عبارة عن كتابة منطق البرنامج باستخدام نصوص جمل بدلا من أشكال رموز خريطة التدفق .

يعتبر أى معالج نصوص أداة كتابة الكود الشكلى Pseudocode ، ويمنح معالج النصوص القدرة على إدراج ونقل النصوص أو حذفها ، ويعد الكود الشكلى أسرع من الهياكل التدفقية اليدوية لأنه لا يحتاج إلى رسم ، علاوة على ذلك فهناك مبرمجون كثيرون يسهل عليهم تحويل منطق الكود الشكلى إلى كود البرنامج النهائى .

مثل خرائط التدفق ليست هناك وسيلة لتعلم أساليب الكود الشكلى إلا بالممارسة ، وأيضا مثل خرائط التدفق يمكن كتابة الكود الشكلى لأى شئ وليس فقط برامج الكمبيوتر .

تستخدم العديد من تعليمات الكتب نموذج الكود الشكلى لشرح خطوات تركيب البرامج أو تجميع الأجزاء .

فيما يلى منطق مشكلة المرتبات فى هيئة كود شكلى Pseudocode الذى يمكن قراءته بواسطة أى شخص حتى إذا كان يجهل رموز خرائط التدفق :

* إدخال ساعات العمل واسم الموظف ووظيفته .
* إذا عمل الموظف عدد ساعات معين أو أقل منه يحصل على أجر بمعدل عدد الساعات مضروبا فى الحد الأدنى للأجور (قرار) .
* إذا عمل الموظف عشر ساعات إضافية عن معدل الساعات الرسمية يأخذ أجر ساعات العمل الرسمية ، مضافا إليه حاصل ضرب عدد الساعات الإضافية مضروبا فى ضعف أجر الساعة العادية (قرار) .
* خصم الضرائب والخصومات الأخرى .
* طباعة شيك الأجر الصافى .

قد يبدو الكود الشكلى سهلا لكنه لن يكون مثل خريطة التدفق فى التوضيح .

الخطوة الثالثة : كتابة البرنامج

بعد تعريف المخرجات وتحديد المنطق اللازم لاستخراج المخرجات يجب الذهاب إلى الكمبيوتر والبدء فى توليد الكود وإنشاء عبارات البرمجة اللازمة لكتابة البرنامج ، وهذا يعنى تعلم لغة برمجة أولا ، وهناك عدة لغات برمجة ، كما لم تعد كتابة البرامج مهمة سهلة للمبتدئين لكن مع الممارسة والتدريب يتمكن من كتابة البرامج وإجادة لغة البرمجة والتفوق فيها .

تستغرق عملية كتابة البرامج وقتا طويلا لكن بعد تعلم البرمجة وممارستها تأخذ عملية البرمجة وقتا أقل خاصة إذا كان التصميم دقيقا ، وتتطلب طبيعة البرمجة تعلم بعض المهارات .

خلاصة

كما هو حال بناء منزل بعد تصميمه فلا يجب كتابة برنامج قبل تصميمه ، فى أوقات كثيرة يتجه المبرمجون إلى لوحة المفاتيح بدون التفكير فى منطق البرنامج ، ويؤدى هذا إلى ظهور برنامج به العديد من الأخطاء والمشكلات ، ويجب التأكد من ملاءمة تصميم البرنامج مع ما يريده المستخدم .

بعد انتهاء تعريف المخرجات يمكن تنظيم منطق البرنامج باستخدام التصميم من أعلى لأسفل Top-Down ، وخرائط التدفق Flowchart ، والكود الشكلى Pseudocode .

يجب ترك التفاصيل جانبا قدر الاستطاعة عند التصميم من أعلى لأسفل Top-Down الذى يعد أداة تحديد جميع التفاصيل التى يحتاجها برنامج .

عامة لا يقوم التصميم من أعلى لأسفل Top-Down بوضع التفاصيل فى ترتيبها المنطق ، ويقوم الكود الشكلى Pseudocode بإملاء منطق البرنامج ، كما يحدد وقت حدوث الأشياء وترتيب الحدوث ووقت توقف الفعل .

البرمجة والأساليب الهيكلية

بعد معرفة الخطوات المطلوب اتخاذها قبل البرمجة وهى : تعريف المخرجات Output ، وبناء منطق البرنامج ، يمكن استخدام أى لغة برمجة متاحة لكتابة البرنامج ، بمعنى أنه سوف تبدأ عملية البرمجة بالفعل بدءا بتحديد لغة البرمجة المستخدمة

لكتابة البرنامج فى البيئة التى يتم تصميم البرنامج لها .

لإنهاء عملية البرمجة بعد كتابة البرنامج يتم اختباره ، وتتضمن الخطوط العريضة للبرمجة ذاتها معرفة :

ما تتطلبه عملية كتابة برنامج ، ومفهوم المحرر Editor ، والفرق بين محرر السطور ومحرر الشاشة ، وأهمية البرمجة الهيكلية Structured Programming ، ومكوناتها ، وخطوات اختبار البرنامج ، والفرق بين المراجعة المكتبية والاختبار التمهيدى Beta ، وأهمية وضرورة الاختبار المتوازى .

استخدام المحرر Editor

المحرر هو أداة كتابة البرامج على جهاز الكمبيوتر فى لغة برمجة ، وهو مثل معالج النصوص العادى من بعض الأوجه ، حيث يمكن كتابة سطور الكود (الشفرة) وتعديلها ونسخها ونقلها ولصقها وحفظها على القرص ، ولهذا يسمى المحرر بمحرر النصوص Text editors .

بخلاف معالج النصوص لا يقوم المحرر بعمل التفاف النص Warp أو انسيابه من سطر إلى آخر ، فعندما تصل إلى نهاية السطر فى معالج النصوص تنتقل الكتابة من نهاية السطر كما ينتقل مؤشر الكتابة كذلك إلى السطر التالى ، أما فى البرامج فلا تكون خاصية التفاف النص موجودة لأنها تغير مفهوم جملة البرمجة .

لغات البرمجة موجزة فى تعبيراتها ، فلا يمكن لعبارات البرمجة أن تتداخل مثل الكتابة العادية ، فى بعض لغات البرمجة تستطيع كتابة أكثر من عبارة فى نفس السطر ، لكن هذا الأسلوب غير مفضل ، حيث يزيد من صعوبة قراءة البرنامج بالنسبة للمبرمج وللآخرين ، مما يتسبب بالتالى فى صعوبة تحديثه وصيانته مستقبلا .

تنقسم برامج المحرر إلى نوعين هما : محرر السطور ، ومحرر الشاشة .

مهما كان نوع المحرر الذى يستخدمه المبرمج فى كتابة البرامج يجب عليه الاعتياد على نوعى برامج المحرر ، حيث يقدم كل منهما ميزات غير موجودة فى الآخر فى ظروف معينة .

١- محرر السطور Line Editor

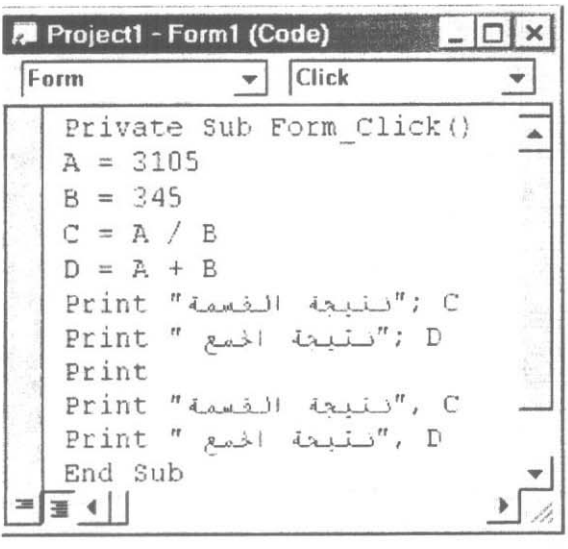
قد يكون محرر السطور قديما وجزءا من الماضى فهو مفيد فقط على الوحدات الطرفية المطورة فى عام ١٩٦٥ .

فى محرر السطور يمكن إدخال نص البرنامج وتعديله ، والفرق بين محرر السطور وبين محرر الشاشة ، أن محرر السطور يسمح بالتعامل مع سطر واحد فقط فى المرة الواحدة ، فهو لا يتيح استخدام مفاتيح الحركة (مفاتيح الأسهم) للتنقل فى الشاشة أو إجراء التعديلات على سطور عديدة مثلما يفعل محرر الشاشة ، بدلا من ذلك يجب أن تحدد لمحرر السطور بدقة السطر المطلوب تعديله ، وبالتالى يمكن فقط التعامل مع هذا السطر ، وبعدها يمكن تحديد سطر آخر لتعديله (التعديل سطرا بسطر) .

أكثر محرر سطور شيوعا لا يزال معتمدا هو محرر إصدار يونكس UNIX المرئى Vi (أو Visual) وهو متاح على الكمبيوتر الشخصى حيث يستخدمه مبرمجو يونكس UNIX الذين انتقلوا للعمل فى بيئة الكمبيوتر الشخصى .

٢- محرر الشاشة Full Screen Editor

محرر الشاشة أسهل فى التعلم والاستخدام من محرر السطور ، ويملك الأدوات المتاحة فى معالجات النصوص ، حيث يتزود بقوائم Menus مع استخدام الفأرة ، وتعليمات مساعدة Help ، وتأتى معظم لغات البرمجة بمحرر شاشة خاص .



هناك عناصر عديدة شائعة فى محررات الشاشة فإطار التحرير الكبير فى منتصف الشاشة هو مكان إدخال النص وتعديله فى البرنامج .

أثناء العمل فى محرر الشاشة تقوم بكتابة نص البرنامج وتعديله ، ويمكن استخدام مفاتيح الأسهم لتحريك المؤشر فى أنحاء الشاشة ، كما يمكن إدراج نص وحذفه كما فى معالج النصوص ، وحفظ البرامج ، وتحميلها من القرص وإليه ، أو البحث عن نص معين ، واستبداله ، أو تحريك كتل من النصوص ونسخها .

يستخدم محرر الشاشة القوائم عبر قمة الشاشة مما يجعل المبرمج غير مضطر لحفظ الأوامر بل يختارها من القوائم ، فقائمة ملف File تختارها بنقر الفأرة ، أو الضغط على مفتاحى ALT +F معا مما يظهر خيارات قائمة ملف File لانتقاء أحد خياراتها (بتحريك المؤشر أو النقر بالفأرة) .

إحدى ميزات دمج المحرر Editor مع المترجم Compiler والمفسر Interpreter هى سهولة الاستخدام فقبل هذا الدمج كان المحرر يحفظ الملف ثم تخرج من المحرر لترجمة البرنامج مما يعنى تعلم مجموعة أخرى من الأوامر ، وإذا كان البرنامج يحتوى على أخطاء يجب العودة مرة أخرى إلى المحرر لتصحيح الأخطاء .

مع دمج المحرر فى لغة البرمجة يمكن استخدام نفس الأوامر من القوائم لإدخال البرنامج وترجمته أو تجميعه ثم تنفيذه ، مما يعنى السهولة وقدرة المبتدئين على التركيز فى محتوى البرنامج ، وعدم القلق بشأن أوامر عمل البرنامج وتنفيذه .

بعد إدخال البرنامج يمكن استخدام القوائم لترجمة البرنامج إلى لغة الآلة وتنفيذه دون الخروج من الإطار ، ويمكن رؤية المخرجات ثم العودة إلى البرنامج الذى أخرج النتائج ، أما إذا كانت المخرجات تحتوى على أخطاء فيمكن العثور عليها وتغيير شفرة البرنامج Source Code .

من أفضل محررات الشاشة المدمجة مع لغة برمجة محرر لغة فيجوال بيزك Visuai Basic ومحرر فيجوال سى Visual C++ بالإضافة إلى اللغات المشابهة ، فهى تمنح قدرات فعالة على التحرير والتعديل والترجمة باستخدام القوائم ، وأدوات برمجة متقدمة مثل أداة تتبع الأخطاء لإصلاحها debugger ، وأداة إيجاز البرنامج وتصميمه Profiler .

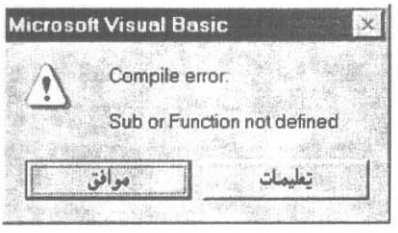
أداة تتبع الأخطاء Debugger

بواسطة هذه الأداة يمكن تنفيذ سطر واحد لاختبار أجزاء البرنامج أثناء تنفيذه ، فبدلا من تخمين سبب الخطأ أثناء تشغيل البرنامج (الذى يكون سريعا للغاية) يمكن فحص المخرجات .

أداة الإيجاز والتقييم Profiler برنامج يقوم بمراقبة تنفيذ البرامج للبحث عن الشفرة المطولة أو البطيئة فى البرنامج لتحسينها من حيث السرعة أو الحجم .

تقوم معظم برامج الترجمة Compilers المستخدمة اليوم بتحسين شفرة البرنامج لتحويلها إلى أسرع لغة آلة ممكنة وأمثلها .

عند كتابة برامج تعمل تحت بيئة ويندوز Windows كمعظم المبرمجين سيكون من المفيد كتابة البرنامج واختباره وفحص الأخطاء داخل بيئة واحدة للبرمجة فى ويندوز Windows مثل فيجوال بيزك Visual Basic .



البرمجة الهيكلية Structured Programming

فى أواخر الستينيات غرقت أقسام البرمجة تحت وطأة البرامج المتراكمة والمتزايدة ، وظهرت الحاجة إلى صيانة البرامج المكتوبة من قبل ، ويجب كتابة برامج سهلة القراءة والصيانة باتباع الدقة ، وبذلك تضمن مستقبل العمل كمبرمج ، كما تستطيع توفير الأموال عند كتابة برامج سهلة الصيانة والتعديل والإصلاح .

عند انتهاء كتابة برنامج يكون قد انتهى العمل فيه فى الوقت الراهن فقط ، فافتراضات هذا البرنامج ووظيفته سوف تتغير مع الوقت ، إن صيانة البرامج القديمة أصبحت عبئا على تطوير برامج جديدة ، فالصيانة تستغرق وقتا .

أثناء أزمة الصيانة فى الستينات بدأ البحث عن طرق جديدة وأساليب حديثة لكتابة البرامج لتعمل بصورة أسرع وأفضل ، وأن تكون سهلة القراءة ليمكن تعديلها

وصيانتها بدون جهد كبير ، وظهرت أساليب البرمجة الهيكلية فى ذلك الوقت .

البرمجة الهيكلية هى فلسفة كتابة البرامج بطريقة مرتبة بدون الكثير من القفزات ، فإذا كان البرنامج قابلا للقراءة بسهولة فهو قابل للتغيير والإصلاح بنفس السهولة .

عرف المبرمجون أن أسلوب الكتابة الواضح هو أمر هام لكنه أصبح واضحا بعد استخدام أساليب البرمجة غير الهيكلية .

بفهم خرائط التدفق Flowcharts والكود الشكلى Pseudocode يمكن معرفة البرمجة الهيكلية باستخدام تلك الأدوات ، وعند تعلم لغة برمجة سوف تفكر بطريقة البرمجة الهيكلية وتعمل بها .

إن البرنامج المكتوب جيدا وسهل القراءة لا يعنى ضرورة اتباع الأسلوب الهيكلى ، فالبرمجة الهيكلية اتجاه محدد فى كتابة البرامج ينتج عنها عادة برامج مكتوبة جيدا وسهلة القراءة ، إذ لا يوجد شئ يعوض خطأ المبرمج الذى يتعجل إنهاء البرنامج بكتابته بطريقة يتصور أنها الأسرع ، ويستمر فى استخدام البرنامج حتى يأتى يوم تصبح فيه تقنيات تعديل البرنامج طويلة وكثيرة إلى درجة تستغرق وقتا أطول من وقت كتابة البرنامج من جديد .

تتضمن البرمجة الهيكلية ثلاث وحدات بناء هى :

- التتابع أو التتالى Sequence .
- القرار Decision ، أو الاختيار Selection .
- الحلقات التكرارية Looping ، أو الإعادة Repetition or Iteration .

طالما كانت لغة البرمجة تدعم تلك الوحدات الثلاث (يجب أن تكون كذلك) فبالإمكان كتابة برامج هيكلية ، أما عكس البرنامج الهيكلى فهو الكود المتشابك Spaghetti Code لبرنامج مبعثر لا يربطه هيكل متماسك ، يحتوى البرنامج غير الهيكلى على تفرعات وتشعبات بدون نظام أو تنسيق .

تتيح معظم لغات البرمجة التفرغ باستخدام عبارة اذهب إلى Go To ، وتخبر هذه العبارة الكمبيوتر أن يذهب إلى مكان فى البرنامج لاستكمال التنفيذ هناك مما قد يفقد تسلسل الأفكار ، تحذر كتب البرمجة من عبارة اذهب إلى Go To وتنصح بالبعد عنها ، وإن كانت هذه العبارة غير سيئة عندما تستخدم بتحفظ ، لكنها يمكن أن تقضى

على قابلية قراءة البرنامج عند الإفراط فى استخدامها .

إن الوحدات الثلاث لبناء البرمجة الهيكلية لا تعمل فقط فى البرامج ، إذ يمكن استخدامها أيضا فى خرائط التدفق Flowcharts ، والكود الشكلى Pseudocode ، أو أى مجموعة من الأوامر ، فهذه الوحدات تعمل على التأكد من تنفيذ البرنامج بدون إفراط فى التفرعات والتشعبات فى كل مكان ، وأن خطوات التنفيذ سهلة التتبع يمكن التحكم فيها .

التتابع أو التتالى Sequence

التتالى هو عبارة عن أمرين أو أكثر يتلو أحدهما الآخر ، ويعتبر مبدأ الأوامر المتتالية أسهل وحدات بناء البرمجة الهيكلية لأنه يمكن تتبع البرنامج بدءا من أول عبارة إلى أخرها .

نظرا لأن برنامج الكمبيوتر لديه القدرة على اتخاذ القرارات أو أداء مهام متكررة فلا يجب أن تحتوى جميع البرامج على عبارات منطقية متتالية فقط ، وهنا يأتى دور القرار أو الاختيار .

القرار أو الاختيار Decision or Selection

فى كل مرة يقوم البرنامج باتخاذ قرار يجب أن يذهب فى أحد اتجاهين ، فالقرار نقطة توقف لمسار البرنامج المنساب فى تتابع ، لكنها نقطة توقف أو تفرع تحت السيطرة ، حيث إن التفرع يجب أن يأتى نتيجة لقرار معين ، ومن ثم فإن البرنامج يجب أن يتخطى الكود غير المطلوب تنفيذه .

القرار بخلاف الاتجاه المستقيم يجعل المبرمج لا يقلق بشأن الكود غير المنفذ فلا يجب العودة لقراءة جزء من البرنامج الذى تخطاه القرار (واعتمادا على البيانات الجديدة قد يكرر البرنامج القرار ويأخذ مسارا مختلفا فى المرة الثانية لكن على أية حال يمكن اعتبار تعتبر الكود الذى لن يتم تنفيذه غير مهم فى الوقت الحالى) .

الحلقات التكرارية (التكرار والإعادة)

Looping (Repetition and Iteration)

أكثر مهام الكمبيوتر أهمية هى التكرار (إعادة تنفيذ سطور معينة من البرنامج) ، قد يقوم الكمبيوتر بتكرار أجزاء من البرنامج آلاف المرات مع بيانات كثيرة لمعالجتها ثم

يمكن للمستخدم تحليل النتائج .

يعتبر التكرار حتميا فى معظم البرامج فنادرا ما تقوم بكتابة برنامج من أوامر متتالية فقط حيث إن المهمة التى يؤديها مثل هذا البرنامج لن توازى وقت أو جهد تصميم البرنامج وكتابته ، وتصبح البرامج فعالة بحق عند تكرار سلسلة من الأوامر أو القرارات المتتابعة .

تقوم الحلقات التكرارية بكسر قاعدة أن خريطة التدفق تنساب دائما إلى أسفل وإلى اليمين ، ويجب الحذر من الحلقة التكرارية اللانهائية Infinite Loop فإذا دخل الكمبيوتر فى إحداها لن يتوقف ، ويصعب إعادة السيطرة على البرنامج بدون إعادة تشغيل الكمبيوتر ، يجب دائما أن تسبق الحلقة التكرارية بعبارة قرار حتى يمكن توقف الحلقة عن التكرار نتيجة للقرار ، وبالتالى يتم تنفيذ باقى البرنامج .

واجهة البرنامج ودقة البرنامج

العامل الأساسى فى شكل تحديد واجهة البرنامج هو : نظام البرمجة المستخدم اعتمادا على إمكاناته ، وقدرات إصداره ، وبيئة نظام التشغيل الذى يعمل عليه ، وتوفر الغالبية العظمى من لغات البرمجة إنشاء واجهات الاستخدام المختلفة .

فى التصميم الجيد للبرنامج يعرف المستخدم عند تشغيل البرنامج نوع البرنامج بالنظر إلى واجهة البرنامج الرئيسية ، وعندما يرى المستخدم برنامجا يعمل يستطيع أن يخمن معظم وظائفه ، ويستطيع أيضا تحديد نظام التشغيل الذى يعمل عليه البرنامج .

نتيجة تعود المستخدمين على الأشكال المألوفة للبرامج الشهيرة فإنهم يتوقعون من المبرمج الجيد إظهار شاشات البرنامج بنفس المفهوم .

تقود معرفة نظام التشغيل الذى تصمم البرامج عليه إلى : تحديد واجهة البرنامج الذى تقوم بتصميمه ، إن واجهة البرنامج تعنى : شكل ظهور شاشة البرنامج ، وطريقة تفاعل الكمبيوتر مع مستخدم البرنامج (استجابة الكمبيوتر للأوامر) .

شكل شاشة البرنامج

إن شكل ظهور البرنامج على شاشة جهاز الكمبيوتر يتحكم فى طريقة البرمجة وأدوات البرمجة المستخدمة لتحقيق هذا الشكل ، وبصفة عامة هناك نوعان من

الواجهات هما : الواجهة النصية ، والواجهة الرسومية .

طريقة تفاعل الكمبيوتر مع الأوامر

أســـاس الـــتفاعل بين المستخدم وأوامر البرنامج المكتوب هو طريقة إعطاء الأوامر وتوفيـــر المدخلات للبرنامج عن طريق المستخدم ، ويعد شريط الأوامر Menu Bar فى برامج الواجهة الرسومية (مع أشرطة الأدوات Tool Bars وشريط الحالة Status Bar ومربعات الحوار Dialog Boxes) الوسيلة الرئيسية لتحقيق هذا التفاعل .

ليـــس المهـــم هو تشكيل واجهة للبرنامج تحتوى على شريط أدوات وشريط قوائم به أوامـــر الـــبرنامج لكـــن المهم أيضا هو توحيد المفهوم اعتمادا على الطرق القياسية لاســـتجابة الـــبرنامج للأوامر مما يستدعى إلمام المبرمج بعموميات تطبيقات الواجهة الرسومية .

الدقة والأخطاء فى البرامج

المشكلة التى تزعج المبرمج هى تجمع الأخطاء التى تظهر فى شفرة برنامج ، ويجب أن يتأكد المبرمجون من خلو البرامج التى يقومون بكتابتها من الأخطاء على الرغم من أن هذا لا يبدو سهلا كما قد يتبادر إلى الذهن .

يسمى خطأ البرنامج علة Bug فعند تحليل مشكلة البرمجة وتقسيمها إلى تعليمات تفصيلية يترك المبرمجون دائما أشياء خاطئة ، أو يصممون أشياء خاطئة ، لذلك فعند تشغيل برنامج تظهر الأخطاء نتيجة لوجود العلل bugs فى الشفرة .

يمثل مصطلح العلة Bug موقفا أنه فى أوائل سنة ١٩٥٠ توقفت طباعة ، وبعد وقت طويل دون الوصول إلى حل لوحظ وجود حشرة (البق) بين أسلاك الطابعة ، مما أدى إلى تعطيلها ، وبمجرد إزالة الحشرة عادت الطابعة إلى العمل ، وأصبحت هذه الحشرة فى تاريخ الكمبيوتر أول عطل Bug للجهاز .

إن كشف العلل (الإصلاح) Debugging هى العملية التى يقوم من خلالها المبرمج بإزالة أخطاء البرنامج ، فأثناء كتابة برنامج يقوم المبرمج بتشغيله فى وضعه غير النهائى (عند إمكانية تشغيله) لتصيد أكبر قدر من الأخطاء الممكنة وعزلها عن البرنامج النهائى ، وعادة فإن غالبية الأخطاء تظهر بعد انتهاء كتابة البرنامج ، ودائما

يفشل المبرمجون المبتدئون فى إدراك توغل الأخطاء فى شفرة البرنامج .
اعتمادا على طول البرنامج فإن الوقت الذى يأخذه المبرمج لإصلاح الأعطال والمشاكل يماثل تقريبا وقت كتابة البرنامج الأصلى ، وقد يصعب العثور على بعض الأخطاء أحيانا .

اكتشاف وتصحيح الأخطاء

لا بـــد من تجربة البرامج قبل تسويقها ، وعند تجربة برنامج من البرامج بتشغيله قد تظهر بعض الأخطاء المحتملة فى بناء البرنامج ، أو فى صيغة تركيب جمل البرنامج ، أو فى هجاء كلمات برمجة البرنامج ، أو فى منطق البرنامج نفسه ، أو فى استخدام أدوات ومعدات غير موجودة فى الجهاز .
قد يتم اكتشاف أخطاء البرنامج بمراجعته سريعا ، لكن قد لا تظهر بعض الأخطاء إلا بعـــد فترة طويلة من التشغيل وتكرار التجربة ، وقد لا تكون الأخطاء ظاهرة بسهولة بسبب تعقد بناء البرنامج ، أو عيب فى بناء حالة من الحالات التى يندر استخدامها .
هناك ثلاث فئات من العلل Bugs هما : الأخطاء الشكلية Syntax Errors ، والأخطاء المنطقية Logical Errors ، وأخطاء التشغيل Runtime Errors (التى يمكن اعتبارها من الأخطاء المنطقية) .

- الأخطاء الشكلية Syntax error يسهل العثور عليها لأنها عادة تكون عبارة عن أوامر لغات برمجة غير صحيحة الحروف (الهجاء) ، أو مشكلات قواعد نحوية تتعلق بطريقة استخدام لغة البرمجة مثل : الأخطاء نحوية فى تركيب جمل ، أو استخدام كلمة بهجاء غير صحيح ، أو استخدام كلمات محجوزة كأسماء لمتغيرات ، أو عدم الإعلان عن متغيرات وثوابت أو غير ذلك من تركيب الجملة بنسيان جزء أو إهمال مسافة أو علامة علامات تركيب اللغة .
- تظهر الأخطاء المنطقية Logic Errors عندما يكون البرنامج سليما من ناحية بناء الجملة لكنه يقوم بعمل شئ لم يكن مفروضا أن يؤديه ، وتتوقف الأخطاء المنطقية على منطق البرنامج نفسه وتركيبه والشروط فى بناء البرنامج .
- النوع الآخر من الأخطاء هو خطأ التشغيل Runtime Error ، وتظهر أخطاء

وقت التشغيل Runtime نتيجة لخطأ منطقى لأن المبرمج فشل فى التوقع ، وبالتالى لم يتعامل مع المشكلة المحتملة ، مثل تخطى قيود التخزين ، والقسمة على الصفر ، أو التدفق الزائد للبيانات ، أو عدم وجود طابعة أو غيرها ، إذ يمكن أن يظهر خطأ وقت التشغيل Runtime error عند محاولة برنامج الكتابة على قرص دون التأكد من إغلاق مدخل مشغل القرص ، وأيضا التأكد من أن القرص موجود داخل المحرك ، كما يظهر خطأ وقت التشغيل Runtime error أيضا على سبيل المثال إذا قام البرنامج بالقسمة على صفر (لأن القسمة على صفر غير معرفة حسابيا) ، وعند التمرس فى البرمجة تزداد قدرة فهم وتوقع والتعامل مع مشكلات التشغيل Runtime error المحتملة وحلها والتغلب عليها .

تحتوى لغات البرمجة على أدوات لاكتشاف الخطأ فى قواعد الجمل ، وتركيب جمل البرمجة خلال التفسير أو الترجمة ، كما أن معرفة الأخطاء العملية متاحة عند تجربة البرنامج مرات عديدة تحت ظروف بيانات مختلفة .

تعد مراجعة البرنامج أيضا وسيلة هامة من وسائل اكتشاف الأخطاء فى البرنامج خاصة الأخطاء المنطقية التى يصعب على مصرف اللغة (المترجم) اكتشافها ، ولتجنب حدوث الأخطاء يتم :

- مراجعة تصميم البرنامج .
- استخدام الوحدات النمطية القصيرة .
- فحص البرنامج أثناء إنشائه ، وتتبع منطق البرنامج .
- استخدام شفرات عزل الأخطاء .
- استخدام أسماء واضحة للمتغيرات .
- الالتزام بمنهجية وقياسية استخدام الأسماء والمتغيرات .
- إضافة تعليقات على معظم أجزاء البرنامج .

بالرغم من الالتزام قدر الإمكان بتعليمات اللغة ، ومحاولة تجنب الأخطاء ، فقد تحدث الأخطاء التى يجب التخلص منها .

توقف البرنامج لسبب ما هو حدث غير متوقع أثناء تشغيل البرنامج ، ولا تستطيع لغة البرمجة تجاهله فمثلا قد تكون الطابعة خالية من الورق ، كما يمكن أن يكون محرك

الأقراص المرنة خاليا من القرص المرن ، أو أن يكون القرص غير مجهز ، ويحدث خطأ التشغيل كلما نفذت لغة البرمجة جملة لا يمكنها إتمامها لأى سبب من الأسباب .
توفر لغات البرمجة أدوات التصحيح المختلفة التى يمكن استخدامها لتتبع أخطاء ، أو منطق ، أو خطوات تنفيذ البرامج ، وتصحيحها .
تتيح لغات البرمجة أيضا كتابة برمجيات روتينات خاصة تسمى : معالجات الأخطاء للتعامل مع أخطاء التشغيل ، حيث تبين للبرنامج كيفية المتابعة عند حدوث خطأ .
عـــادة يتم استعمال معالج الأخطاء لمعالجة أحداث خارجية تؤثر على سير البرنامج مثل بعض المشاكل المحتملة التالية التى يمكن اصطيادها بمعالجات الأخطاء :
تعطـــل محرك أقراص ، أو أقراص غير مجهزة ، أو باب محرك مفتوح ، أو قطاع قـــرص تالف ، أو مشاكل الطابعة لطابعات لا تعمل ، أو خالية من الورق ، أو غير مـــتوفرة ، أو أخطـــاء الفائض مثل كثرة معلومات الطابعة أو الرسم ، أو أخطاء نفاذ الذاكـــرة ، ونقص فى مساحة التطبيق أو موارد النظام ، أو مشاكل نقل البيانات ، أو فى حافظـــة النظام ، أو الأخطاء المنطقية فى بناء الجمل ، أو فى المنطق لم يكتشفها المصرف (المترجم) .
تستعمل الجمل البرمجية لكشف خطأ التشغيل أو الجمل المتوقع حدوث خطأ بسببها ، وتضـــبط ، أو تنشط فخا من خلال إبلاغ لغة البرمجة أين يذهب منطق البرنامج عند حدوث خطأ .
عادة يتألف معالج الأخطاء من قسمين :
أ- القسم الأول : بنية قرار تعرض رسالة أو تضبط خاصية بناء على الخطأ .
ب- القسم الثانى : جملة تعيد التحكم أو تتفرع بعد تصحيح سبب الخطأ .
تعتبر البرامج سهلة الكتابة نوعا ما لكن إصلاح البرامج مسألة أخرى فتحديد مكان أخطاء البرنامج يعد صعبا ، ولحسن الحظ يقوم معظم منتجى لغات البرمجة بتوفير أدوات تحديد الأخطاء لجعل البرمجة أكثر سهولة ، وعلى الرغم من أن برنامج المترجم يحدد أخطاء النص فإن الأخطاء المنطقية تأخذ وقتا كثيرا فى تحديدها ، ويجب تحديد المشكلة قبل قيام المستخدمين بتحديدها .
تطورت وزادت أدوات تحديد الأخطاء ، وتعد أدوات ذات أهمية قصوى للمبرمجين ،

فعند استخدام بيئات برمجة مثل Visual basic تقوم بتوفير خصائص أكثر قوة لمكتشف الأخطاء مثال :

* تحليل المتغيرات فى وقت التشغيل (فى إطار المباشر Immediate) وعرض تلك المتغيرات فى إطار منفصل عن المخرجات ، ويمكن أيضا تحليل محتويات قيم المتحكم وبذلك يمكن بسهولة فحص أى من القيم التى تم تخزينها فى مربع أو خاصية زر أمر .

* تغيير محتويات المتغيرات أثناء تنفيذ البرنامج حتى يعمل باقى البرنامج كما لو أن الكود قد قام بإسناد تلك القيم .

* وضع نقاط الإيقاف خلال البرنامج .

* وضع متغيرات مراقبة تقوم بوقف تنفيذ البرنامج عندما تستقبل المتغيرات قيمة معينة أو مجموعة من القيم .

* تخطى العبارات التى لا تريد تنفيذها أثناء عملية تحديد الأخطاء .

على الرغم من عدم استطاعة أدوات تحديد أماكن الأخطاء المنطقية بمفردها فهى تجعل المهمة أسهل .

من المفاهيم القوية لمكتشف الأخطاء هو تفاعله مع البرنامج أثناء لحظة نقطة الإيقاف فعندما يصل برنامج إلى نقطة إيقاف فإن جميع القيم التى تم حسابها لا تزال موجودة لذلك يمكن عرض المتغيرات لرؤية نتائجها ، ويمكن تغيير قيمة متغير فى منتصف تنفيذ برنامج ومراقبة كيفية قيام التنفيذ بتمثيل ذلك التغيير .

اختبار البرامج Testing

لا يعنى انتهاء كتابة الكود الفعلى للبرنامج الانتهاء من هذا البرنامج ، يجب تتبع أخطاء البرنامج لاستبعاد أى احتمال للأخطاء الممكن حدوثها ، وحتى لا يكتشف مستخدم البرنامج تلك الأخطاء ، لذا يجب فحص البرنامج واختياره بدقة وحرص ، فيما يلى الخطوات النموذجية التى يجب أن يتبعها المبرمجون قبل توزيع الإصدار الأخير من برنامجهم على المستخدمين :

١- أداء الاختبار المكتبى Desk Checking .

٢- عمل اختبار تمهيدى Beta .

٣- مقارنة نتائج الاختبار التمهيدى Beta بنتائج الاختبار المتوازى للنظام القديم .

الاختبار المكتبى Desk Check

يقوم معظم المبرمجين بعمل سلسلة من الاختبارات المكتبية على برامجهم ، وهى عبارة عن اختبار البرنامج أثناء عمله على الكمبيوتر بشتى أنواع البيانات المحتملة ، وذلك لإيجاد نقاط الضعف وأخطاء الكود ، وأثناء تلك العملية يقوم المبرمج بتجربة قيم غير متوقعة مثل إدخال رقم أو قيمة خاطئة ، وأيضا تجربة جميع خيارات البرنامج المتاحة واستخدام تركيبات مختلفة لرؤية ما يحدث فى كل المواقف .

الاختبار التمهيدى Beta Testing

عندما ينتهى الاختبار المكتبى ويصبح المبرمج على ثقة كاملة من صحة البرنامج ودقة المخرجات ، على المبرمج أن يختار مجموعة من المستخدمين لتجربة البرنامج ، وتسمى تلك المرحلة بالاختبار التمهيدى Beta ، وكلما زاد عدد المختبرين للبرنامج كلما زاد احتمال العثور على أخطاء ، إذ أن المستخدمين يجدون عادة أشياء لم يفكر فيها المبرمجون أثناء كتابة البرنامج .

تقوم الشركات بدعوة الجمهور للمساعدة فى الاختبار التمهيدى Beta بطرح نسخ مجانية تجريبية Beta من برامجها ونظم تشغيلها قبل طرح الإصدار النهائى بوقت طويل ، وتكون تلك النسخ متاحة للتحميل من الإنترنت حيث تعطى المستخدمين نظرة على المنتج وبالتالى مساعدة الشركات حيث يقوم هؤلاء المختبرون بإبلاغها بأية أخطاء يجدونها ، وبينما يزداد عدد جمهور المختبرين يزيد أيضا وقت الاختبار .

المشكلة هى أن البرامج اليوم معقدة للغاية فمثلا قد يعمل مائة مبرمج لإنتاج برنامج ولا تظهر فيه الأخطاء قبل مرحلة طويلة من التشغيل .

الاختبار المتوازى Parallel Testing

لا يجب على المستخدم أن يهجر النظام القديم تماما وينتقل إلى البرنامج فورا بل يجب إجراء الاختبار المتوازى فمثلا : إذا كتبت برنامجا لحساب الرواتب بدلا من نظام

يدوى قديم يجب الاستمرار فى استخدام النظام القديم جنبا إلى جنب مع البرنامج .
صحيح أن ذلك معناه أن حساب الرواتب سوف يستغرق وقتا أطول كل مرة ، لكن يمكن مقارنة نتائج البرنامج مع اليدوية للتأكد من تطابقها ، وبعد عدة مرات يستطيع المستخدم الشعور بالثقة حيال استخدام البرنامج بدون مطابقة ، وخلال فترة الاختبار قد يضطر المبرمجون لعمل تغييرات على البرنامج .
يجب توقع التغييرات الضرورية ، ولا يجب أن يكون ذلك مثارا لهز الثقة فى قدرة المبرمج على البرمجة ، أو الشعور بخيبة الأمل ، إذ نادرا ما يكتب المبرمج برنامجه صحيحا من أول مرة ، فلابد من عدة محاولات لجعله صحيحا .
على الرغم من ذلك لا تضمن الاختبارات الشاملة المذكورة برنامجا مثاليا ، فبعض الأخطاء لا يمكن أن تظهر بعد فترة من الوقت والاستخدام ، وكلما زادت الاختبارات كلما قلت فرصة ظهور الأخطاء لاحقا .

رفع كفاءة الأداء

يعنى رفع كفاءة الأداء عدة أمور منها تحسين شكل البرنامج ، وتنظيم منطق البرنامج ، وتصحيح الأخطاء وتكرار تجربته ، وزيادة سرعة عمله ، والاستخدام الأمثل لموارد الكمبيوتر ، وتصغير حجم البرنامج ، واستخدام بيئة البرمجة المناسبة ، ومراجعة الشفرة لتقليل العمليات والنصوص ، وفهم إمكانيات مصرف اللغة والاستخدام الأمثل له ، وتقدير وجود واستخدام معدات ومكونات مختلفة فى الكمبيوتر مثل المعالج ونظام التشغيل والماسح والطابعة ووحدات الإدخال المختلفة للاستفادة منها ، وغيرها من العوامل التى تزيد من كفاءة البرنامج .
تتوفر الأدوات الكثيرة التى يمكن استخدامها لرفع كفاءة البرامج ، كما تتوفر أدوات قياس كفاءة أداء البرامج أيضا Benchmark تفحص شفرة البرنامج وتعطى تقريرا يساعد على فهم ومراجعة شفرة البرامج وأوجه القصور فى بنائها .
أثناء تطوير مهارات البرمجة واستخدام لغاتها المختلفة تصادف بعض الأدوات التى تلزمك فى وضع شفرات البرامج وتحسين الأداء ، و تؤدى أدوات البرمجة عدة أمور أكثر من لغات البرمجة نفسها .

إحدى أدوات البرمجة أداة تقييم الأداء Profiler التى تقوم بتحليل أجزاء من البرنامج وتحديد الأجزاء البطيئة ، وتحتوى أغلب لغات البرمجة على أداة تقييم للأداء profiler أو يقوم المبرمجون بإنتاج واحدة .

يستخدم أيضا تحكم الإصدار Version Control لتتبع إصدارات البرامج ، ويقوم تحكم الإصدار بتتبع إصدارات البرامج الموزعة وتتبع جميع شفرات المصدر التى تخرج للمستخدمين ، فيمكن للمبرمج استخدام برنامج تحكم الإصدار التعامل مع ملفات مشروع ، وتتضمن لغات البرمجة مثل فيجوال بيزك Visual Basic تحكم إصدار باسم Source Safe كخيار يستخدم لحفظ برنامج مصدر .

تأتى أى لغة للبرمجة فى ويندوز بمتطلباتها مصاحبة لها كأدوات لمساعدة مبرمجى ويندوز ، ومن هذه الأدوات محرر الموارد Resource Editor ، موارد ويندوز عبارة عن أى شئ يستخدم فى ويندوز فقد تكون رمزا أو جملة أو نص أو صورة أو قائمة أو مربع حوار ، وعند العمل مع لغات البرمجة يتعامل المبرمج مع تلك الموارد وتظهر الموارد Resources فى مشروع التطبيق بأحد الملفات التى تنتهى بامتداد اسم الملف RES .

هناك طرق عديدة لاستخدام موارد تطبيقات ويندوز ، ويساعدك فى ذلك أداة محرر الموارد Resource Editor الموجود فى مجموعة Visual Studio ليقوم بالإضافة والحذف أو تحرير الموارد بمشروع التطبيق .

اعتقد الناس لسنوات أن مهمة المبرمجين سوف تختفى مع مرور الزمن ولكن ما حدث أن الاحتياج إلى المبرمجين قد ازداد مع زيادة أهمية البرمجة .

فى السبعينيات كانت نظم معلومات الإدارة Management Information Systems (MIS) جل جميع متطلبات الحاسب ، وكان من المفترض أن تستخدمها جميع الشركات وتصبح هى البيانات التى تحتاجها الشركات فى متناول كل مستخدم هذا النوع من تنقية البيانات كان لابد وأن يكون فعالا حتى أن البرامج العادية والأكثر تخصصا تصبح غير مطلوبة لكن من الواضح أن وعد MIS لم يكن فقط مبالغ فيه لكنه أيضا لم يصل أبدا إلى حيز التنفيذ .

فى أواخر الثمانينيات ظهرت أداة هندسة البرامج بمساعدة الحاسب Computer

Aided Software Engineering (CASE) التى كانت سوف تحل محل المبرمجين وبدلا من وجود مصممى برامج يتقنون لغة أو أكثر من لغات البرمجة يتم الاعتماد على مبرمجين يجيدون استخدام أدوات CASE .

تعتبر CASE أداة لإنتاج البرامج فقط بدلا من مساعدة المبرمجين فى كتابة البرامج ، وتعمل أدوات CASE على مساعدة موظفى أقسام معالجة البيانات فى إنشاء برامج تبدأ عند مستوى التصميم الأول كما يمكن لمحلل النظم Systems Analyst استخدام CASE من بداية طلب البرنامج إلى أن يصل البرنامج إلى مرحلة الإنتاج .

أداة CASE هى برنامج ضخم يستخدمه محلل النظم لتصميم المخرجات الأولية ، وتعريف البيانات ، وتعريف الفكرة المنطقية ، وتقوم بعض برامج CASE برسم خرائط التدفق Flowcharts من خلال وصف خريطة التدفق التى يدخلها محلل النظم وإنتاج البرامج أيضا ، وغالبا ما تقوم CASE بإنتاج كود يعتمد على تعريف الفكرة المنطقية لمحلل النظم لكن لابد من تدخل المبرمجين لإنجاز البرامج والتأكد من نجاح المشروع .

لقد وعدت CASE بإحداث ثروة فى بيئة البرمجة وتقليل الزمن والموارد اللازمة لإنتاج برنامج كامل (أغلب أهداف البرمجة الحديثة هى الاتقاء بسرعة التطور وسهولة الصيانة) ، وعلى أية حال فلم تخرج وعود CASE إلى حيز التنفيذ وحتى إذا كانت قد حققت بعض النجاح فمازال عليها أن تقدم التحسينات المألوفة فى تطوير البرامج المتقدمة .

إن منتجات CASE فى الثمانينيات لم تكن بالأدوات السيئة لكن المشكلات التى نتجت عنها كانت ترجع إلى أن CASE تساعد محللى النظم والمبرمجين على أداء مهامهم بسرعة أكبر على الرغم من عدم صحة تلك المهام ، وقد عانت الوسائل فى الفترة قبل البرمجة الموجهة للكائنات من صعوبة الصيانة ومشكلات التوثيق فلم تتمكن CASE من تقليل المشكلات الملازمة التى تتضمنها البرمجة الأخرى غير التابعة للبرمجة الشيئية OOP .

يعتبر المبرمج أداة CASE أحد البرامج التى تساعد على تصميم البرامج وكتابتها وتعتبر مناسبة فى تناول التفاصيل الدقيقة فى تطوير النظام ليتمكن من إنجاز متطلبات المستخدمين .

فى الآونة الأخيرة أصبحت أدوات المبرمجين غاية فى التعقيد حتى أن البعض يعتقد

أن تكنولوجيا المعالجة مثل تطبيقات Visual Basic سوف تزداد قوة حتى تصبح البرمجة تقريبا مجرد إجابة على مجموعة أسئلة .

لغة النمذجة الموحدة Unified Modeling Language (UML) تقدم تعريفا موحدا لنمذجة برنامج ، وعلى ذلك فإن المبرمج أو الشركة التى تقوم بعمل نموذج لأحد البرامج يمكنها أن تشارك الشركات الأخرى التى تكتب برامج مماثلة لهذا النموذج ، ولا يمكن اعتبار النماذج أكوادا لكنها تعريفات للتطبيقات .

إن لغة النمذجة UML يعتبر أساسيا ومفيدا لمصممى قاعدة البيانات ، ولمن يكتبون تطبيقات قاعدة البيانات للمشاركة فى محتويات كل قاعدة من قواعد البيانات ، ولنقل تلك المحتويات بين الحاسبات ، وتصميمات البرامج ، وتقديم الفوائد التالية :

* إعادة استخدام Reuse التصميم .
* إمكان الاستخدام فى نظم متعددة Tool Interoperability .
* التطوير من خلال فريق Team development .
* إدارة موارد البيانات Data resource management .
* تتبع الاعتماد Dependency tracking .

مما يعمل على تحسين البرمجة ، ويتيح استخدام التصميم التابع لنظام معين مع النظم الأخرى للارتقاء بإنتاجية المبرمج .

يستمر المبرمجون فى الارتقاء بمهاراتهم لمواكبة التغيرات التقنية فى لغات البرمجة ونظم التشغيل والعتاد ، وكلما زادت معرفته ومعلوماته بالبرمجة يجب مشاركة الآخرين فى هذه المعلومات من خلال التدريب والاستشارة والقراءة و المكتبات وشبكة الإنترنت ، فالتدريب لن يتوقف أبدا لمواكبة الأحداث والتطورات فى صناعة المعلومات للاطلاع على كل ما هو جديد وبحث موضوعات جيدة وجديدة ، والتعمق بصورة أكبر فى لغات البرمجة .

بعد إجادة البرمجة لابد من الاستمرار فى الدراسة حيث إن التزود بالمعلومات فى عالم الحاسب لا يقل أهمية عن ممارسة المجال للارتفاع بمستوى البرمجة والتقليل من المشكلات التى تواجه صناعة البرامج .