بسم الله الرحمن الرحيم

جامعة السّودان المفتوحة

برنامج الحاسوب

هندسة البرمجيات (1)

رمز المقرر ورقمه: هند **1029**

تأليف: د. أحمد أحمد شعبان دسوقي

أ . د.السيد محمود عبد الحميد الربيعي

التحكيم العلمي: د. عوض الكريم محمد يوسف

تصميم تعليمي:أ. عبد الباسط محمد شريف

التدقيق اللغوي: أ. الخاتم عبد الرحمن

التصميم الفني: منى عثمان أحمد النقة

منشورات جامعة السودان المفتوحة، الطبعة الأولى 2006

مقدمة المقرر

الحمد لله ، الذي خلق الإنسان ، علم البيان ، وله الحمد أن رفـــع الـــذين آمنـــوا والذين أوتوا العلم درجات.

والصلاة والسلام على رسول الله المبعوث معلماً للناس أجمعين ، ورضـــي الله عن صحابته أجمعين ، ومن تبعهم بإحسان إلى يوم الدين.

عزيزي الدارس ،،

بين يديك كتاب "مقدمة في هندسة البرمجيات".

هندسة البرمجيات هو التخصص الذي يهتم بعمليات بناء البرمجيات التي تلبي متطلبات المستخدمين ، وذلك من خلال تطبيق النظريات والمعرفة بأسلوب فعال وبكفاية موثوقة ، وتدخل الطرق المستخدمة في هندسة البرمجيات في جميع أطوار دورة حياة البرمجيات والتي تشتمل : تحليل المتطلبات ، وضع المواصفات ، التصميم ، البناء ، الاختبار والتكامل ، التثبيت والتسليم ، والصيانة.

والفرق بين هندسة البرمجيات وعلوم الحاسوب هو أن الأخير يهتم بالنظريات والأساسيات ، في حين أن هندسة البرمجيات تهتم بالوسائل التطبيقية فـــي إنتـــاج وتشغيـــل هذه البرمجيات، وتكمن صعوبة هندسة البرمجيات في أنه علم جديد تتنوع فيه الآراء والنظريات في كل طور من أطوار دورة الحياة للبرمجية ،هذا إضافة إلى احتياجه للعديد من الفرعيات المتعلقة بفروع العلوم الأخرى كإدارة المشاريع ودراسات الجدوى.

ولقد أعدت جامعة السودان المفتوحة برنامج البكالوريوس في علوم الحاسب لتلبية الحاجة المتنامية للمتخصصين في مثل هذا المجال المتطور ، وقد أشتمل هذا البرنامج على منهج خاص لهندسة البرمجيات باللغة العربية ليكون أمام خريجي هذا البرنامج فرص عملهم مهندسي برمجيات ، والتي من المتوقع زيادة الطلب عليها

خلال السنوات القليلة المقبلة خاصة في مجالات تطبيقات النظم الضخمة سوأ كانت مجالات التطبيق حكومية أم خاصة.

وبمراجعتنا لصفحات الويب والكتب التي صدرت باللغة العربية في هذا المجال خرجنا بالانطباعات التالية :

- أن المقالات المتناثرة عبر الويب والمكتوبة في الفرعيات المختلفة يكتبها غير متخصصين في المجال (عبارة عن اجتهادات شخصية) بصورة غير واضحة إضافة إلى أنها غير موثقة.
- أن الكتب العربية المكتوبة في هذا المجال قليلة العدد ، ومعظمها عبارة عن تراجم حرفية لكتب إنجليزية صادرة في هذا المجال، ولم يحدد فيها بالضبط فئات تلقي هذه الكتب ومتطلباتها.
- لم تصغ الكتب بصور وحدات بحيث تكون صالحة لأن تكون مناهج دراسية متكاملة إضافة لعدم اشتمالها على أنماط الأسئلة التدريبية للمراجعة على فصولها المختلفة ، وكذلك عدم وجود اهتمام بمعاني المصطلحات.

وفي إعدادنا لهذا المنهج وجدنا إنه من الضروري أن ننحى منحى آخر مختلف تماما لتلافي ما ورد ذكره واضعين في اعتبارنا منهجية تحتوي على النقاط الأساسية التالية :

- أن نتجنب التراجم والصياغة الحرفية من الكتب باللغة الإنجليزية، ولكن يتم تفهم الفرعية من (المراجع المتعددة الحديثة الموثقة ومن الدوريات العلمية العالمية في هذا المجال) ومن ثم يتم صياغتها بطريقة مفهومة.
- أن تتم صياغة الفرعيات في صورة وحدات منفصلة وفقا للمعايير الدولية في إعداد المناهج على أن تزيّل الوحدات بالتمارين والأسئلة وإجابات بعضها للمراجعة.
- أن يتم صياغة المنهج (مجموعة الوحدات) بصورة متدرجة ومتسلسلة

ليناسب العديد من فئات القراء المتخصصون وغير المتخصصين دون الاحتياج إلى متطلبات مناهج أخرى.

- اشتمال كل وحدة من الوحدات على مسرد للمصطلحات الخاصة بها حيث تفتقر الكتب والمقالات المنشورة لمعاني هذه المصطلحات باللغة العربية.

ويهدف هذا الكتاب إلى تزويد الطالب بالمعرفة اللازمة والتهيئة لمستقبل مهني في مجالات هندسة البرمجيات وإدارة مشاريع البرمجة وما يدخل فيها من تطوير للنظم وتكاملها في مجالات متعددة ومنها تقنيات المعلومات ، حيث إن بناء النظم أصبح يشبه مجال الهندسة إذ تدخل الإدارة السليمة والمنهجية في صلب بناء النظم الناجحة.

ويشمل هذا الكتاب الوحدات التالية :

الوحدة الأولى : وعنوانها : "مفاهيم أساسية في هندسة البرمجيات" ، وفيها نحاول إعطاء الدارس فكرة واضحة ودقيقة حول مجموعة من المفاهيم لهندسة البرمجيات ، والتحديات الرئيسية التي تواجها ، والمسئولية المهنية والأخلاقية لمهندسي البرمجيات.

الوحدة الثانية : وعنوانها : " دورة حياة (عمليات) البرمجيات" وهي دراسة تفصيلية للمراحل المختلفة لدورة حيات البرمجيات.

الوحدة الثالثة : وعنوانها : "نماذج تطوير عمليات البرمجيات" ، وفيها يتم تعريف نماذج تطوير عمليات البرمجيات والدور الذي تقوم به ، إضافة إلى الشرح التفصيلي لهذه النماذج ، ودراسة الفروق المختلفة بينها ، ودراسة مدى اعتماد بعضها على بعض.

الوحدة الرابعة : وعنوانها : " هندسة متطلبات البرمجيات" ، وفيها يتم توضيح كل من مفهوم هندسة المتطلبات ، والأنشطة (المراحل) التي تشملها عملية هندسة المتطلبات ومدخلاتها ومخرجاتها.

الوحدة الخامسة : وعنوانها : "التحقق والمصادقة على صحة البرمجيات" ، وفيها تم بيان الفرق بين التحقق من صحة البرمجية وبين المصادقة عليها ، ومعرفة طبيعة الأخطاء الأكثر شيوعا وانتشارا في البرامج ، وتوضيح الفحوصات والآليات والطرق المختلفة التي يتم من خلالها إجراء عمليتي التحقق والمصادقة.

الوحدة السادسة : وعنوانها : " مبادئ تقدير تكلفة البرمجيات" ، وفيها تم توضيح أهمية التقدير الدقيق لتكلفة المشاريع البرمجية ، والعوامل التي تؤثر على دقة هذا التقدير ، مع شرح وافٍ لأهم الطرق والمنهجيات المختلفة التي تستخدم لتقدير الجهد والتكاليف وفترة الجدولة الزمنية ، والمتغيرات الأساسية التي تدخل في ذلك ، مع ذكر بعض الإرشادات والأفكار المفيدة لتقدير تكلفة المشروعات البرمجية.

الوحدة السابعة : وعنوانها : " النماذج الخوارزمية لتقدير البرمجيات" ، وفيها تم تعريف نماذج التقدير الخوارزمية والشكل العام لها ، وبيان الصعوبات التي تعاني منها ، وكيفية قياس مدى دقتها ، وبيان الأسس التي يتم على أساسها تقويمها شرح وافٍ لأهم أنواعها (نموذج بوتنام ، ونموذج كوكومو) ، حيث نستعرض كل نموذج في شكله الرياضي نوضح كيفية استنتاج ثوابته من الجداول الخاصة به. هذا بالإضافة إلى شرح تفصيلي للشروط التي يجب أن تؤخذ في الاعتبار عند تقويم أدوات تقدير تكلفة المشاريع البرمجية.

الوحدة الثامنة : وعنوانها : "خطوات تقدير تكلفة المشروعات البرمجية" ، وفيها تم نناقش خطوات تقدير تكلفة المشروعات البرمجية ، حيث يتم استعراض الغرض من كل خطوة ، والخطوات التنفيذية لتحقيق الغرض منها ، والدخل والخرج الخاص بها ، والمسئولين القائمين على تنفيذها.

الوحدة التاسعة : وعنوانها : " مقاييس جودة البرمجيات وتأمينها" ، وفيها يتم تعريف الجودة وسماتها المختلفة ، والأنشطة الخاصة بتأمين جودة البرمجيات ، بالإضافة إلى شرح وافٍ لأهم مقاييس البرمجيات وتصنيفاتها المختلفة.

نأمل أن تقدم هذه الوحدات صورة واضحة عن هندسة البرمجيات ، وان تتيح لك عزيزي الدارس الفرصة لاكتساب المفاهيم والمهارات مما يمكن لك مستقبلاً فهم بقية المقررات التي تعتمد على هذا المنهج.

عزيزي الدارس، لدى قراءتك لهذا الكتاب أرجو أن لا يغيب عن ذهنك أن مادة هذا الكتاب هي الحد الأدنى من المعرفة في موضوعه، الذي يسمح بإثرائه على الدوام، بالرجوع إلى القراءات المساعدة والاطلاع عليها. وينصح عند قراءة مادة الكتاب محاولة الإجابة عن الأسئلة التقويمية، والعودة إلى النص للتأكد من صحة الإجابة، كما ينصح بعدم الرجوع إلى إجابات التدريبات إلا بعد محاولة حلها أولاً.

هنيئا لك عزيزي الدارس بهذا المقرر، وأنا واثق من أنك ستجده ممتعا ومفيدا، وسأوجز لك فيما يلي أهم الأهداف العامة منه.

الأهداف العامة لهذا المقرر

بعد فراغك من دراسة هذا المقرر ينبغي أن تكون قادراً على أن:

- تعرِّف هندسة البرمجيات.
- تصف دور هندسة البرمجيات في هندسة وبناء البرمجيات.
- تشرح الفرق بين علم هندسة البرمجيات وعلوم الحاسبات الآلية.
- تعدد عمليات البرمجيات.
- تصف هيكلاً عاماً لعمليات البرمجيات.
- تشرح دورة حياة البرمجيات.
- تدون أهم مراحل تطوير البرمجيات وأهمية كل منها في عملية التطوير.
- تحدد الفروق المختلفة بين نماذج تطوير عمليات البرمجيات.
- تختار النموذج المناسب لتطوير منتج برمجي طبقاً لمعطيات هذا المنتج.
- تعدد أهم مميزات وعيوب كل نموذج من النماذج التي تم تناولها.
- تشرح مفهوم هندسة المتطلبات.
- تصف هيكلاً عاماً لأنشطة (مراحل) عملية هندسة البرمجيات.
- تصف طبيعة الأخطاء الأكثر شيوعا وانتشارا في البرامج.
- تشرح الفرق بين التحقق من البرمجية والمصادقة عليها.
- تُعرِّف عملية تقدير تكلفة المشاريع البرمجية.
- تعدد العوامل الرئيسة التي تتحكم في عملية تقدير تكلفة المشاريع البرمجية.
- تصف النماذج الخوارزمية المستخدمة في تقدير تكلفة المشاريع البرمجية.
- تقيم نماذج تقدير تكلفة المشاريع البرمجية من خلال استخدام أهم المعايير الخاصة بذلك.
- تقيس مدى دقة تقدير تكلفة المشاريع البرمجية لأي نموذج خوارزمي.
- تعرف أدوات تقدير تكلفة المشاريع البرمجية مع ذكر أهم سماتها.
- تذكر أهم خطوات تقدير تكلفة المشاريع البرمجية مع ذكر مدخلات ومخرجات كل منها.

- تعرِّف جودة البرمجيات.
- تعدد أهم السمات الأساسية التي يجب أخذها في الاعتبار لإنتاج برمجيات ذات جودة عالية.
- تصف بإيجاز أهم السمات التي تحدد مدى جودة البرمجيات.

وختاماً نرجو أن نكون قد وفقنا في تقديم إضافة جديدة لمكتبة جامعة السودان المفتوحة من خلال هذا الكتاب ونسأل الله تعالى أن يكون هذا العمل خالصا لوجهه وأن يكون في ميزان حسناتنا وأن ينتفع به الجميع في وطننا العربي ، ويغفر لنا ما سهونا عنه وعجزنا عن إدراكه.

ونحن نرحب بكل نقد بناء يسهم في تطوير هذا العمل إلى الصورة الأفضل، والله نسأل الأجر والثواب وآخر دعوانا أن الحمد لله رب العالمين إنه نعم المولى ونعم النصير.

وفيما يلي قائمة بعناوين الوحدات الواردة في هذا المقرر مع أرقام صفحاتها.

محتويـــات المقـــرر


الوحدة الأولى
مفاهيم أساسية في
هندسة البرمجيات

محتويات الوحدة

مقدمة

تمهيد

عزيزي الدارس ،،

سنتعرف من خلال هذه الوحدة إلى مفاهيم أساسية في هندسة البرمجيات ، والتي تعد الأساس لهذه المادة .

ما هي هندسة البرمجيات ، سنقوم بتوضيح شقي المصطلح ، الشق الهندسي والشق البرمجي ، في القسم الثاني من الوحدة نتناول المهام النموذجية لهندسة البرمجيات ، وهي مهام تهتم بجميع أطوار دورة حياة تطوير البرمجيات بدءاً من المراحل الأولى لتحليل النظام حتى صيانته ، القسم الثالث يتناول مجموعة من المبادئ الأساسية التي تتميز بها هندسة البرمجيات، ثم تنتقل بك الوحدة إلى القسم الرابع وهي أنواع البرمجيات وهنا نستطيع أن نميز نوعان رئيسان: برمجيات النظام ، البرمجيات التطبيقية ، في القسم الخامس نتناول أدوات هندسة البرمجيات بمساعدة الحاسب ، وهي طائفة واسعة من الأنواع المختلفة من البرامج التي تستخدم في مساندة الأنشطة الخاصة بالعمليات البرمجية ، القسم السادس يتناول التحديات الرئيسة التي تواجه هندسة البرمجيات في القرن الحادي والعشرين والتي تتمثل في التحدي المتعلق بالموروث ، تحدي تحديد عدم التجانس ، تحدي المخرجات ، القسم السابع يتناول مهنة البرمجة وفريق العمل الخاص بهذه المهنة المكون من محللو النظم ومصممو النظم ثم المبرمجون ، القسم الثامن والأخير يعالج موضوع المسئولية المهنية والأخلاقية لمهندسي البرمجيات.

تجد عزيزي الدارس فى ثنايا الوحدة بعض التدريبات التي يساعدك تنفيذها على فهم محتوى المادة هذا بالإضافة إلى أسئلة التقويم الذاتي التي تهدف إلى ترسيخ الفهم وتعزيزه لديك و تعميقه .

أهداف الوحدة

عزيزي الدارس،

بنهاية دراسة هذه الوحدة ينبغي أنّ تكون قادراً على أن :

- تعرِّف هندسة البرمجيات.
- صف دور هندسة البرمجيات في هندسة وبناء البرمجيات.
- تشرح الفرق بين علم هندسة البرمجيات وعلوم الحاسبات الآلية.
- تصف العلاقة بين هندسة البرمجيات وهندسة النظم..
- تعدد عمليات البرمجيات.
- تعرِّف أدوات هندسة البرمجيات بمساعدة الحاسب الآلي.
- تشرح أهمية أدوات هندسة البرمجيات بمساعدة الحاسب الآلي في بناء البرمجيات.
- تسرد المهام النموذجية لهندسة البرمجيات.
- تعدد مبادئ هندسة البرمجيات.
- تسمي أهم أنواع البرمجيات.
- تشرح التحديات الرئيسة التي تواجه هندسة البرمجيات.
- تصف المسئولية المهنية والأخلاقية لمهندسي البرمجيات والعمل بها.

توطئة

في عصرنا الحالي والذي يُعد عصر المعلوماتية يعتمد الإنسان في معظم أمور حياته سواء في بيته أو في مكان عمله أو في الأماكن العامة على التعامل مع العديد من الأجهزة والمعدات التي تعتمد في عملها على البرمجيات والتي تعتمد بدورها على أنظمة الحاسب الآلي ، لذا أصبح من المهم تصميم هذه الأنظمة وبرامجها بأقل التكاليف من خلال التوزيع السليم للموارد المتاحة والتصميم الجيد لهذه الأنظمة والبرامج بحيث تعمل بكفاءة وموثوقية عالية وبطريقة ذات فعالية لتعلب دورا فعالا في الاقتصاد الوطني والعالمي ، ومن هنا ظهر مفهوم هندسة البرمجيات.

وقد استخدمت هندسة البرمجيات كمفهوم نظري من حين لآخر في أواخر الخمسينات وبداية الستينات من القرن الماضي، أما الاستخدام الرسمي لها فكان في مؤتمر عُقد من قِبل اللجة العلمية في منظمة حلف شمال الأطلسي عام 1968م حول البرمجيات ، وقد أخذ هذا المصطلح في الانتشار منذ ذلك الحين ولاقى اهتماماً متزايداً في نواحي مختلفة.

قبل ظهور هندسة البرمجيات كان تطوير البرمجيات يفتقر إلى الكثير من المعايير المتعارف عليها اليوم مما نتج عنه ذلك فشل العديد من المشاريع البرمجية ، أو زيادة الوقت اللازم لإنجازها ، أو صعوبة في صيانتها. وحتى تلك النظم البرمجية التي تم تصميمها وإنجازها كانت غالبا ما تعاني من نقص في الموثوقية وظهور العديد من الأخطاء غير المتوقعة إضافة لدعم وثائقي ضعيف وعدم تلبيتها للشروط والمتطلبات التي صُممت لأجلها بشكل كامل. ولحل هذه المشاكل والصعوبات كان من اللازم إعادة النظر وتقويم جميع الجوانب المتعلقة بتصميم وتطوير البرمجيات مما أدى إلى ظهور هندسة البرمجيات مع بداية عام 1968م نتيجة للنقاشات حول ما عرف في ذلك الوقت "بأزمة البرمجيات" حيث تم لأول مرة الإشارة إلى طرق تصميم وتطوير البرامج، وقد ساعد على ذلك ــ حينها ــ ظهور الجيل الثالث من أجهزة

الحاسب الآلي القوية ، حيث إن قوتها جعلت النظم البرمجية غير الممكنة سابقاً ممكنة ، ونتج عن ذلك أنظمة ضخمة وكبيرة وأعقد من الأنظمة السابقة ، مما أدى إلى توسع مجالات استخدام الحاسوب في معظم مجالات الحياة.

1. ما هي هندسة البرمجيات؟ (What is Software Engineering?)

يوجد العديد من التعريفات التي تم اقتراحها من العديد من المهتمين بهندسة البرمجيات ويمكن إجمال هذه التعريفات في أن "**هندسة البرمجيات**" هي : علم يهتم ببناء الأنظمة البرمجية الكبيرة والمعقدة بواسطة فريق من مهندسي البرمجيات باتباع طرق ومبادئ هندسية منظمة وصحيحة واستخدام وسائل وأدوات وتقنيات تجعل من تصميم وتطوير وبناء واختبار هذه النظم عملية منظمة يمكن تكرارها في حدود الإمكانيات المتاحة مادياً وبشرياً وزمنياً ، وذلك من خلال دراسة دورة حياة النظام مع إعطاء أشكالاً متعددة لعمليات تصميمه وتطويره وبنائه واختباره بحيث يكون المنتج البرمجي النهائي على درجة عالية من الجودة والموثوقية وقليل التكاليف بقدر الإمكان مع تسليمه للعميل في الوقت المناسب على أن يعمل بكفاءة على أجهزة الحاسب الآلي المختلفة.

وبصفة عامة ، يهتم علم هندسة البرمجيات بجميع نواحي إنتاج نظم البرمجيات التي تلبي متطلبات المستخدمين تحت قيود معينة ، وذلك من خلال تطبيق النظريات والمعرفة بأسلوب فعال وبكفاءة موثوقة، وتدخل الطرق المستخدمة في هندسة البرمجيات في جميع أطوار دورة حياة نظم البرمجيات من المراحل الأولى لتحليل النظام حتى صيانته مروراً بعمليات وضع المواصفات ، والتصميم ، والبناء ، والفحص ، والتشغيل. كما يوظف علم هندسة البرمجيات الطرق والأساليب والعمليات والقياسات الهندسية لتحقيق الأهداف وإتمام الفائدة من الأدوات المعدة لإدارة عمليات تطوير البرمجيات ، مع الاهتمام بالجودة والتحكم في النوعية.

وبالتدقيق في مصطلح " هندسة البرمجيات " نجده يحتوي على شقين :
الشق الأول : هندسي : وفيه يطبق المهندسون النظريات والأساليب والأدوات الملائمة

بفعالية للحصول على حلول مثالية للمشاكل التي تواجههم مع التقيد بالقيود التنظيمية والمالية التي تحيط بالمشكلة ، ودائماً ما يحاولون اكتشاف الحلول للمشاكل حتى عندما لا توجد نظريات أو أساليب مطبقة لتدعمهم ، وذلك من خلال تبنيهم أسلوب تصنيفي منظم لعملهم مع التركيز على اختيار الطريقة الأكثر ملائمة لتنفيذ النظام البرمجي طبقاً للمواصفات المطلوبة والطريقة الأكثر ابتكاراً.

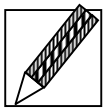
الشق الثاني : فهو برمجي : وفيه يتم الاهتمام بجميع مظاهر إنتاج البرمجيات ، حيث إن هندسة البرمجيات لا تهتم فقط بالعملية الفنية لتطوير البرمجيات ولكن تهتم أيضاً بالنشاطات الأخرى مثل إدارة مشروع البرمجيات ، وتطوير الأدوات والأساليب والنظريات لدعم إنتاج البرمجيات.

وبناءاً على ما سبق يتضح أن دور هندسة البرمجيات هو : **تطبيق منهج مرتب وقابل للقياس لعمليات تطوير وتشغيل وصيانة البرمجيات ، أي تطبيق الهندسة على البرمجيات** ، وهو ذات التعريف الذي وضعه معهد المهندسين الكهربائيين والإلكترونيين **(IEEE)** لهندسة البرمجيات ، وذلك من خلال الإجابة على العديد من الأسئلة التي تخص هندسة وتطوير البرمجيات ، والتي من أهمها :

- ما هي الطرق والمبادئ الهندسية المنظمة والصحيحة التي يمكن تطبيقها على بناء البرنامج؟.
- ما هي الوسائل والأدوات والتقنيات التي تجعل من تصميم وتطوير وبناء واختبار البرنامج عملية منظمة يمكن تكرارها في حدود الإمكانيات المتاحة مادياً وبشرياً وزمنياً؟
- كيف نتمكن اقتصاديا من بناء برنامج موثوق يعمل بكفاءة ليس على جهاز حاسب واحد بل على العديد من أجهزة الحاسب المختلفة؟.
- ما هي القياسات والإجراءات التي يجب إجراؤها لضمان جودة المنتج البرمجي؟.
- كيف سيتم دعم البرنامج على المدى الطويل عندما يطلب المستخدم إجراء تعديلات أو تحسينات عليه؟.

- ما هو المنهج الواجب اتباعه لدراسة تقنيات الإدارة اللازمة لتخطيط المشاريع البرمجية وتنظيمها واستخدامه كآلية للتحكم في المشروع بشكل يقود إلى تسليم منتج برمجي عالي الجودة قابل للتشغيل في الوقت المحدد؟.

تدريب(1)

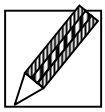


ما هو الفرق بين علم هندسة البرمجيات وعلوم الحاسبات الآلية؟

تدريب (2)

ما هو الفرق بين هندسة البرمجيات وهندسة النظم؟

تدريب (3)



ما هي عمليات البرمجيات؟

2. المهام النموذجية لهندسة البرمجيات

(Typical Software Engineering Tasks) :

هناك مجموعة من المهام النموذجية التي تتعلق بهندسة البرمجيات يجب أن تتبع في تطوير المشاريع البرمجية من قِبل فريق متكامل من مهندسي البرمجيات ، وهذه المهام هي كالتالي :

• تحليل المشكلة.	• تحديد المتطلبات.	• تصميم البرمجية.
• الترميز (عملية البرمجة).	• اختبار وتكاملية الشفرة البرمجية.	
• التثبيت وتوزيع البرمجية.	• توثيق البرمجية.	• صيانة البرمجية.
• التدريب	• تقدير الموارد	• إدارة المشروع

وبنظرة ثاقبة للمهام المذكورة أعلاه نجد أن هندسة البرمجيات تهتم بجميع أطـــوار دورة حياة تطوير البرمجيات **(Software Life Cycle)** بدءاً من المراحل الأولى لتحليـــل النظام حتى صيانته، وسوف نتناول أطوار دورة حياة تطـــوير البرمجيـــات بالتفصـــيل لاحقاً ، ولكن سوف تغطي فكرة بسيطة هنا عن أهم الخطوات المتبعة (الأطـــوار) فـــي تطوير البرمجيات من وجهة نظر هندسة البرمجيات :

- مبدئيا لا بد من التعرف على متطلبات النظام فإذا كان هناك زبون أو عدة زبائن فإنه لا بد من حدوث مقابلة بين الزبائن ومتخصصي دراسة وتحليل المتطلبات في فريـــق التطوير ، وإن لم يكن هناك زبون محدد ولكن هناك متطلبات لفئة عامة من الزبائن فإن دراسة متطلبات واحتياجات السوق يتم عملها وهذه الطريقـــة مناســـبة لتطـــوير برمجيات عامة.

- بعد دراسة وتحليل المتطلبات والانتهاء من كتابة وثائق التحليل يتم الانتقال إلى وضع التصميم الأمثل لترجمتها إلى شفرة برمجيـــة قابلـــة للتنفيـــذ ، وذلـــك مـــن خـــلال المصممين في فريق العمل ولا بد أن يكونوا ملمين إلماماً كاملاً بطـــرق التصـــميم المختلفة والفروق المختلفة بينها ولا بد أن يكون لديهم إلمام بخواص النظم الموجودة مثل أنظمة التشغيل ، ونظم المعلومات المختلفة المتاحة ، وواجهات التطبيق المختلفة الرسومية والبرمجيات المساعدة التي يمكن أن تساعد في عملية الترميز (البرمجة).

- بعد الانتهاء من تخطيط وثائق التصميم يتم الانتقال إلى توزيع المهام على المبرمجين في فريق العمل لتحويل تلك المخططات وترجمتها إلى شفرة برمجية قابلـــة للتنفيـــذ باستخدام إحدى لغـــات البرمجـــة عاليـــة المســـتوى **(High Level Programming Languages)** ، ومن ثم تبدأ عملية التكامل للنظام وهـــي عمليـــة تجميـــع المهـــام البرمجية بعد ترميزها عن طريق فريق التطوير مع برمجيات جاهزة تخدم في نفـــس المشروع.

- بعد الانتهاء من مرحلة برمجة وتكامل النظام تبدأ مرحلة اختبار النظام البرمجي من حيث الأخطاء البرمجية ، وكذلك من حيث مطابقته للمواصفات المطلوبة وجودة الأداء. ومصطلح جودة البرمجية **(Quality Assurance)** يتعلق بعملية تطوير المنتج البرمجي والطريقة التي تم تطويره بها بحيث تكون وفقا لمعايير مقاييس الجودة والتي يتم وضعها من خلال مجموعة مراقبة الجودة من فريق التطوير بعد تحديد متطلبات المشروع البرمجي مباشرة وقبل البدء في التنفيذ الفعلي للمشروع، وتتم عملية مراقبة الجودة عند الانتهاء من أي جزئية أو مكون من المشروع حيث يتم اختباره وفقاً للمعايير المتفق عليها سابقاً ، وعند وجود أي خلل أو خطأ فعلى الفريق المختص إصلاحه.

- عملية التوثيق ليست مرحلة منفصلة ولكنها تتم بالتوازي مع المراحل السابقة ، وهي لا تشمل عملية التعليق على الشفرة البرمجية فقط ولكنها تشمل العديد من العمليات مثل كتابة ملفات المساعدة ، وكتيب المستخدم ، وكتيب التدريب، وكتيب المرجع ، وكتيبات التثبيت ولا غرابة أن يكون حجم ما يكتب من سطور لتوثيق النظام يزيد على مرتين من عدد سطور النظام نفسه.

- بعد عمليات التحليل ، والتصميم ، والبرمجة ، والاختبار والتوثيق ، لا بد أن توزع هذه البرمجيات وتثبت على أجهزة العملاء.

- ومع بداية إصدار وتوزيع المنتج البرمجي تبدأ مرحلة الصيانة، ومصطلح الصيانة هنا يصف النشاطات التي تتم بعد إصدار المنتج البرمجي والتي تشمل عملية تصحيح الأخطاء ، وتطوير البرمجية لتعمل مع أوساط جديدة وحاسبات جديدة وأنظمة تشغيل جديدة ، وزيادة فاعلية البرمجية، وفي كثير من الحالات تكون الصيانة هي الأكثر كلفة من ناحية المصاريف والوقت في دورة حياة تطوير البرمجية.

وبالطبع فإن كل العمليات السابقة لا بد من إدارتها من خلال إدارة واعية وعلى دراية كاملة بجميع مراحل تطوير مشـــاريع البرمجيـــات ، حيـــث تمثـــل عمليـــة إدارة المشروع النشاط المهم والحرج لتطوير البرمجيات.

3.مبادئ هندسة البرمجيات (Software Engineering Principles)

لهندسة البرمجيات أهمية كبيرة ، إذ أنها تساعد في زيادة المـــردود لشـــركات النظم البرمجية ، ومستخدمي البرمجيات، من خلال التوزيع السليم للموارد المتـــوفرة ، والتصميم الجيد لهذه البرمجيات مما يقلل من تكاليف وزمن إنتاجها ، مع زيادة وتحسين مستوى جودتها وموثوقيتها ، ويتم ذلك من خلال تطبيق مجموعة من المبادئ الأساسية التي تتميز بها والتي من أهمها ما يلي :

- **الدقة والصورية (Rigor And Formality)** : يمكن الحصول على منـــتج موثـــوق ومضبوط الكلفة من خلال الدقة في عملية الإنتاج وتوجد درجات متنوعة للدقة أعلاهـــا الصورية التي تتطلب أن تكون المنتجات مصممة ومقيمة بطريقة رياضـــية، وبالتـــالي تقتضي الدقة الصورية ولكن العكس غير صحيح إذ يمكن أن نكون دقيقين بطرق غيـــر صورية (رياضية). يحتاج أي عمل لتنفيذه إلى خطوات وإذا نفذنا هذه الخطوات اعتماداً على خبرات وتجارب ونظريات عندها ستزداد الدقة، فإذا لم تتـــوفر هـــذه الاعتمـــادات (الخبرات والتجارب والنظريات) نعتمد عند ذلك على التمثيل الرياضـــي وهـــذه هـــي الصورية واللغات البرمجية أدوات صورية للتعبير عن وظائف ثم تحـــول المترجمـــات اللغة الصورية إلى لغة الآلة.

- **فصل الاهتمامات (Separation of Concern)** : يعتمد هذا المبدأ على اتباع سياسة " فرق تسد" حيث يتم تقسيم المشكلة إلى مشاكل أصغر غير مرتبطة ببعضها الـــبعض أو يكون ارتباطها صغيراً ، ثم التركيز على حل كل مســـألة علـــى حـــدة. ويـــتم فصـــل الاهتمامات حسب الطرق التالية :

- **حسب الزمن :** جدولة الأعمال وفق محور الزمن.
- **حسب وجهات النظر:** مثلاً عند تحليل متطلبات نظام برمجي ، قد يكون من المفيد التركيز على المعطيات التي تتعامل معها البرمجية من جهة والتركيز على الوظائف والتحكمات التي تقوم بها البرمجية من جهة أخرى.
- **حسب الخواص والمواصفات :** يتضمن معالجة المواصفات المطلوبة من البرمجية على حدة، فعلى سبيل المثال إذا كان المطلوب بناء نظام برمجي فعال وصحيح ، فيتم أولاً تصميمه بطريقة متأنية ومنظمة تضمن صحته ثم يتم التغيير في النظام لتحسين فعاليته.
- **حسب الأقسام : وهو عبارة عن** فصل التعامل مع الأقسام المكونة للنظام البرمجي كل على حدة.

▪ **التجزئة (Modularity)** : وهي تقسيم النظام البرمجي المعقد إلى أقسام أبسط تسمى كتلاً ، ويهدف مبدأ التجزئة في هندسة البرمجيات إلى :

- القدرة على تجزيء النظام البرمجي إلى كتل (التجزيء التنازلي (الشجري) للنظام البرمجي).
- القدرة على تركيب النظام البرمجي من كتل (التركيب التصاعدي للنظام).
- القدرة على فهم الكتل كلاً على حدة لتعديلها (الاستقلالية).

يجب أن يتحقق في الكتل :

- تماسك قوي داخلي للكتل (الكتلة مبنية بشكل منطقي لتحقيق هدف محدد).
- ترابط ضعيف بين الكتل (الاستقلالية).

▪ **التجريد (Abstraction)** : وهو تحديد المظاهر المهمة وتجاهل تفاصيلها أثناء دراسة مسالة معينة (فصل الشيء المهم عن الشيء غير المهم في المسألة) ، وهي نظرة نسبية مرتبطة بالهدف الذي نعمل عليه.

- **توقع التغيير (Anticipation of Change)** : وهو إحصاء الأماكن التي يمكن أن يتم تعديل عليها في البرمجيات ، ومن أكثر هذه الأماكن تغييراً :

- **الخوارزميات :** يمكن أن تكون هناك عدة خوارزميات تقوم بنفس الوظيفة (الفرز مثلاً) وبالتالي فاستبدال إحدى هذه الخوارزميات بأخرى أفضل منها، هو أمر محتمل ، ولذلك يفضل كتابة الخوارزمية في وحدة برمجية منفصلة.
- **تمثيل المعطيات :** يتغير أداء البرنامج بتغير بنية المعطيات التي يستخدمها.
- **الآلات التجريدية:** يمكن اعتبار البرامج التي نكتبها يتم تنفيذها على آلات تجريدية تتفهم التعليمات والأوامر المكتوبة بلغة عالية المستوى، أي أن لكل لغة عالية المستوى آلة تجريدية تتفهم تعليمات اللغة وتنفذها بشكل تجريدى وبعيد عن العتاد الصلب.
- **الوسط الاجتماعي :** التغير في الوسط الاجتماعي يؤدي إلى تغيير النظام البرمجي المرافق (مثلاً تغير العملة يؤدي إلى تعديل النظم البرمجية البنكية الموجودة) .
- الطرفيات : يتعلق هذا النوع بالبرمجيات التي تحتاج للتعامل مع طرفيات خاصة كأنظمة التحكم. وبالتالي تغيير الطرفيات يتطلب تغيير النظم البرمجية المرافقة.

- **العمومية (Generality)** : يجب علينا عند طرح مشكلة أو مسألة أن نحاول إيجاد مسألة أعم تحوي المسألة المطروحة، لأنه قد يكون حل المسألة الأعم أسهل من المسألة الأصلية، ويمكن أن يكون الحل العام قابلاً لإعادة الاستخدام. وكما يمكن أن يكون الحل العام موجوداً في المكتبات البرمجية الجاهزة، ويعتمد هذا المبدأ عندما يكون الهدف من إنتاج النظام البرمجي تسويقياً.
- **إعادة الاستخدام (Reuse) :**

يشير هذا المصطلح إلى تطوير برمجية معتمده على استخدام أجزاء جاهزة تم إعدادها وفقا لمعايير الجودة حيث تم اختبارها من قبل مما يزيد من إنتاجية المشروع وتوفير المزيد من الوقت، ويمكن النظر لهذا المصطلح – أيضاً – على أنه استخدام ما هو متاح فعليا لتحقيق ما هو مطلوب.

4. أنواع البرمجيات

البرمجيات تحيط بنا في كل مكان : في المناطق الصناعية ، وفي الاستعمالات الخاصة ، وفي أنظمة الاتصالات ، وفي أنظمة النقل ، وفي مجالات أخرى، فالبرامج تأتي مختلفة في أشكالها وأحجامها، فمنها ما يدمج في الهواتف المحمولة ، ومنها ما تقوم بتصميم مركبات الفضاء. أما عن تصنيف البرمجيات ، نستطيع أن نميّز نوعان رئيسيان:

- **برمجيات النظام (System Software)** : هي البرامج التي تمثل الأدوات التي تساعد في بناء أو دعم البرمجيات التطبيقية. وتتميز برمجيات الأنظمة بالتفاعل الكثيف مع عتاد الحاسب الآلي وباستعمال كثيف والتشارك في الموارد من قِبل المستخدمين ، مثل : أنظمة التشغيل ، ومترجمات اللغات ، وخلافه.
- **البرمجيات التطبيقية (Application Software)** : هي البرامج التي تساعد في تأدية بعض المهام المفيدة أو الممتعة بشكل مباشر وأمثلة ذلك : الألعاب ، برامج الصرافة الآلية (**ATMs**) ، برامج التحكم في الطائرة وبرامج البريد الإلكتروني وبرامج معالجات النصوص ، وبرامج الجدولة.

وضمن تصنيف البرمجيات التطبيقية ، من المفيد أن نميز أصناف البرمجيات التّالية:

- **برمجيات الألعاب (Games Software).**
- **برمجيات نظم المعلومات (Information Systems Software)** : وهي من أهم وأوسع مجالات التطبيقات البرمجية ، حيث تقوم هذه النظم بالاتصال بقاعدة بيانات كبيرة أو أكثر تحتوي على كميات ضخمة من البيانات والمعلومات الخاصة بهذه النظم وبطريقة تسهل الحصول على المعلومات واتخاذ القرارات ، والتفاعل مع المستخدم لإتمام عملياته بيسر وموثوقية كبيرة ، ومن هذه الأنظمة على سبيل المثال : نظام حجز مقعد بالخطوط الجوية ، نظام المعاملات البنكية ونظام شئون الموظفين والرواتب ، خلافه من الأنظمة.

- **برمجيات نظم الوقت الحقيقي (Real – Time Systems Software)** : هي الأنظمة التي يجب أن تعطي إجابة ضمن شروط زمنية محددة ، وتتألف مكونات برمجيات الزمن الحقيقي من : مجسات لجمع وتصفية البيانات من المحيط الخارجي ، ومحوّل يقوم بتحويل البيانات إلى الشكل الذي يقبله النظام ، بالإضافة إلى مكوِّن تحكم وإخراج إلى المحيط الخارجي ، ومراقب تحكم ينسق بين جميع هذه المكونات لتستمر المحافظة على الاستجابة بالزمن الحقيقي. ومثال على ذلك برمجيات التحكم في مراكز الطاقة ، والأقمار الصناعية ، والعمليات الصناعية المختلفة.
- **البرمجيات المدمجة بالأجهزة (Embedded Software)** : تقوم هذه البرمجيات بأداء وظائف خاصة ضمن الأجهزة الذكية من خلال برمجة ذاكرات للقراءة فقط بهذه البرمجيات وتركيبها ضمن هذه الأجهزة. مثال على هذه الوظائف : التحكم في سرعة محرك الغسالة الكهربائية ، وكذلك التحكم في اختيار برامج تشغيلها.
- **البرمجيات المكتبية (Office Software)** : وهي برمجيات موجه لأجهزة الحاسبات الشخصية **(PC Computer)** لأتمتة الأعمال المكتبية مثل : مجموعة برمجيات ميكروسوفت أوفس كبرنامج معالجة الكلمات وتنسيق النصوص ، وبرنامج الجداول الحسابية ، وتقديم العروض التقدمية ، وخلافه . وكذلك برمجيات الإنترنت والبريد الإلكتروني ، وخلافه من هذه البرمجيات.
- **البرمجيات العلمية والهندسية (Engineering and Scientific Software)** : تتميز هذه الأنظمة بمحاكاة الأنظمة وتعتمد على الخوارزميات العددية ، وتتراوح تطبيقاتها من إيجاد جذور معادلة من الدرجة الثانية إلى تحليل إحداثيات مركبات الفضاء وإجراء التفاعلات النووية .
- **برمجيات الذكاء الاصطناعي (Artificial Intelligence Software)** : تعتمد برمجيات الذكاء الاصطناعي على مجموعة من الحقائق **(Facts)** والعلاقات **(Relationships)** . وتقوم هذه البرمجيات بتحديد الحل المناسب من خلال الاستنتاج والاستدلال المنطقي **(Logical Deductions)** وبكفاءة عالية ، حيث يتم تمثيل العلاقات

بين الأشياء وتجميعها وتنظيمها للوصول إلى استنتاج منطقي للحقائق التي تمثلها تلك العلاقات. ومثال ذلك برمجيات التعرُّف على الأشكال (الصور والصوت).

- **البرمجيات المغلفة (Shrink-wrap Software)** :

يطلق هذا المصطلح (البرمجيات المغلفة) على البرمجيات التطبيقية المُعدة للتوزيع التجاري **(Commercial Software Packages)** (مثل : ميكروسوفت أوفس). ويتم تحويل البرمجيات التطبيقية المُعدة للاستخدام الداخلي **(In-House Application Software)** إلى برمجيات تطبيقية معُدة للتوزيع التجاري (برمجيات مغلفة) بإضافة العديد من الوحدات البرمجية الجديدة الصغيرة إليها مثل برمجية حماية الشيفرة ، وبرمجية التثبيت وبرمجية الرخص ، ..... وخلافه ، وبعد ذلك يتم وضعها كحزمة برمجية تطبيقية تجارية قابلة للتثبيت على أقراص مدمجة مغلفة تُعد للبيع التجاري. ومن أهم الخصائص أو الاعتبارات الهامة التي يجب أن تؤخذ في الاعتبار عند تطوير هذه البرمجيات ما يلي :

①سهولة التكيف مع منصات تشغيل الحاسب الآلي (Multi-Platform Resilience) :

لا بد أن يراعى في عملية التطوير للحزمة بأن تكون صالحة للعمل على منصات التشغيل المختلفة للحاسب **(Platform Independent)** ، وإذا كانت غير موائمة لمنصات معينه ، يجب العمل على تكييفها للعمل معها عن طريق إضافة وحدات برمجية تقوم بهذا الغرض.

②الخيارات الإقليمية وخيارات اللغة (Localization) :

يجب أن تتوافق هذه البرمجيات مع منصات التشغيل ذات اللغات المتعددة **(Multi-Languages Platforms)** عن طريق إضافة وحدة برمجية **(Multi-Languages Service Pack)** يتم من خلالها اختيار الخيارات الإقليمية وخيارات اللغة بسهولة تامة مما يساعد على انتشار البرمجية وبالتالي على تسويقها.

③إدارة عملية الترقيع (Patch Management) :

وهي العملية الخاصة بتنسيق عمليات التحديثات اللازمة للإصدارات ، وكيفية

توزيع الرقيعات **(Patches)** إلى العملاء لسد الثغرات التي تنشأ في الحزمة أثناء عمليات التشغيل (مثل الرقيعات الأمنية التي توزيعها شركة ميكروسوفت لسد الثغرات الأمنية التي تكتشف بعد توزيع نظام التشغيل) ، وكيفية حماية العملاء من أي فقد للبيانات أو بيانات التشكيل ، وكيفية تحديث الحزمة.

④رخصة الاستخدام وعملية التنشيط للمنتج (Activation License Control and Product) :

عملية تنشيط المنتج البرمجي هي عبارة عن إجرائية للتحقق من صلاحية رخصة استخدامه **(License Activation Procedure)** لمنع استخدام المنتج لغير المرخص لهم ، لذا يجب تضمين هذه العملية في البرمجيات التجارية للحفاظ على حق الملكية ، ويجب أن تتضمن أيضاً طريق سهلة لعملية التنشيط من قبل العملاء، ويوجد العديد من أنواع رخص التشغيل وهي رخص محددة بوقت معين وتتم لنسخ المنتج التقويمية **(Time Limited Evaluation Edition)** (مثل البرمجيات المشتركة التي توزع من خلال الإنترنت للاستخدام لمدة شهر) ، أو رخص استخدام غير محددة الوقت (مثل رخص التشغيل الخاصة بحزمة ميكروسوفت أوفس).

⑤حماية البرنامج المصدر (Source Code Protection) :

لا بد من حماية برنامج المصدر من العابثين الذين يحاولون الحصول عليه ، وذلك باللجوء إلى وسائل التشفير والحماية المختلفة.

⑥وجود وسائل المساعدة (Help) :

لا بد من احتواء الحزمة على طرق لمساعدة العميل في استخدام البرنامج بمعظم اللغات العالمية المشهورة ، وكذلك إرشادات في حالة حدوث مشاكل **(Troubleshooting)** ، ويجب أيضاً تخصيص مواقع على شبكة الإنترنت للمنتج لتقديم المساعدة للعملاء.

⑦التثبيت (Installation) :

وفقا للأبحاث الحديثة الخاصة بتثبيت البرمجيات وجد أن حوالي 30% من فشل حزم البرمجيات ينشأ أصلا من عدم التثبيت الصحيح ، لذا لا بد من التفكير جديا في

طريقة قوية لعملية التثبيت لتجنب الأخطاء التي تؤدي إلى فشل التثبيت ، ولا بد من الأخذ في الاعتبار أنواع الحاسبات المختلفة ، والتفكير في العمليات غير العادية في عملية التثبيت كوضع مفاتيح للتسجيل والرخص ، وعملية التنشيط.

أسئلة تقويم ذاتي

1.ما هي البرمجيات المغلفة؟ وكيف يمكنك تحويل البرمجيات التطبيقية المطورة للاستخدام الداخلي لتصبح برمجيات مغلفة.

2.الجدول التالي يحتوي العمود الأول منه على الأنماط المختلفة للبرمجيات والعمود الثاني منه على تعريفات لهذه الأنماط وهي غير مرتبة، أعد بناء الجدول بحيث يكون هناك توافق بين الأنماط والتعريفات.

أنماط البرمجيات	تعريفها
• برمجيات النظام • برمجيات تطبيقية • برمجيات الألعاب • برمجيات نظم الوقت الحقيقي • البرمجيات المدمجة بالأجهزة • البرمجيات المكتبية • البرمجيات العلمية والهندسية • برمجيات الذكاء الاصطناعي	• برمجيات موجهة لأتمتة الأعمال المكتبية. • برمجيات لمحاكاة الأنظمة وتعتمد على الخوارزميات العددية. • برمجيات تعتمد على مجموعة من الحقائق والعلاقات حيث يتم تحديد الحل المناسب من خلال الاستنتاج والاستدلال المنطقي. • برمجيات تعطي إجابة ضمن شروط زمنية محددة. • تقوم هذه البرمجيات بالاتصال بقواعد البيانات بطريقة تسهل الحصول على المعلومات واتخاذ القرارات. • برمجيات تقوم بأداء وظائف خاصة ضمن الأجهزة. • برمجيات للتسلية وزيادة الإدراك والمعرفة. • برمجيات تساعد في بناء أو دعم البرامج التطبيقية.

5.أدوات هندسة البرمجيات بمساعدة الحاسوب

(CASE Tools)

CASE : هي كلمة مؤلفة من أوائل حروف عبارة **(Computer Aided Software Engineering)** (هندسة البرمجيات بمساعدة الحاسب) وهي تشمل طائفة واسعة من الأنواع المختلفة من البرامج التي تُستخدم في مساندة الأنشطة الخاصة بالعمليات البرمجية مثل تحليل المتطلبات وإعداد النماذج التحليلية وتصحيح الأخطاء والاختبار ، وتتوافق جميع الطرق في أيامنا هذه مع تقنية **CASE** ومنها أجهزة التمثيل بالرموز وفي وحدات التحليل المستخدمة في تدقيق نماذج النظام وفق قواعد توثيقية عملية لإنشاء النظام. وقد تشتمل أدوات **CASE** كذلك على معدات الترميز (التكويد) والتي تقوم بصورة آلية بإعداد تكويد للبرامج من واقع نموذج النظام إضافة إلي إعداد الخطوط الإرشادية الخاصة بتنفيذ العملية وتقديم النصح لمهندسي البرامج حول الخطوة التالية التي يجب القيام بها.

وهذا النوع من أدوات **CASE** والذي يهدف إلى إسناد ودعم عملية التحليل والتصميم يسمى في بعض الأحيان بأداة **CASE** العليا ، لأنه يقدم المساندة للمراحل المبكرة من عملية البرمجة وخلافا لذلك فإن أدوات **CASE** المعدة لتقديم المساندة والدعم لعملية التنفيذ والاختيار كأنظمة تصحيح الأخطاء وتحليل البرامج وإعداد متطلبات الاختيارات وأدوات تحرير البرامج تسمى أحياناً بأدوات **CASE** السفلي .

ويمكن القول بأن أدوات هندسة البرمجيات بالاستعانة بالحاسب الآلي هي : عبارة عن برامج مصممة لمساعدة المبرمجين على التغلب على تعقيدات العمليات البرمجية المختلفة والمساعدة على أتمتتها ، وهي تمثل مجموعة من الضوابط والآليات التي تساعد في تحليل ، وتصميم ، ونمذجة وترميز ، واختبار ، وتوثيق البرمجيات. ويُعد برنامج محرر النصوص كمثال وأداة قيمة من أدوات تطوير البرمجيات لتنظيم الملف أو البرنامج ، وكذلك برنامج محرر الرسوم كمثال آخر لتوثيق الرسوم الخاصة بالتصميم

للنظم الكبرى من البرمجيات ، والمثير للجدل أن هذه الوسائل المساعدة لم يتم اكتشافها وتقنينها بصورة كامله ، فكلما ظهرت احتياجات برمجية جديدة أخرى أصبح من الضروري اكتشاف وسائل وأدوات جديدة مناسبة لها.

1.5 ميزات أدوات هندسة البرمجيات بمساعدة الحاسوب (CASE Tools Advantages)

يمكن تلخيص أهم ميزات أدوات هندسة البرمجيات كالتالي :

①**السرعة الفائقة (Increased Speed) :** تتميز أدوات هندسة البرمجيات بالسرعة الفائقة في أتمتة العمليات وتخفيض الوقت اللازم لاستكمال العديد من المهام خاصة التي تتطلب المخططات الرسومية والبنود الخاصة بها. وتقدر التحسينات في الإنتاجية بعد التطبيق في مدى يتراوح من (200 – 300%).

②**الدقة المتزايدة (Increased Accuracy) :** تساعد أدوات هندسة البرمجيات في اكتشاف الأخطاء والذي تُعد خطوة مهمة في إزالة هذه الأخطاء حيث يوفر اكتشاف الأخطاء وإزالتها الوقت والجهد في المراحل المبكرة من إعداد النظام ، و كلما زاد حجم النظام أصبح من الصعب اكتشاف الأخطاء مما يؤدي إلى زيادة الوقت والجهد المطلوب للتنسيق والإدارة بين فرق العمل المختلفة.

③**تخفيض الوقت اللازم للصيانة (Reduced Lifetime Maintenance) :** وكنتيجة للتحليل الجيد، التصميم الجيد، توليد الشفرات آليا، اختبار البرمجيات آليا، التوثيق الآلي، يتحسن أداء النظام وبالتالي فإن الجهد اللازم للصيانة ينخفض إلى حد كبير. وكذلك يمكن توفير العديد من المنابع لتطوير أنظمة جديدة ، وتقوم أدوات مساعدة البرمجيات (برمجيات إعادة الهندسة **Re-engineering Programs**) باكتشاف الأجزاء من البرمجيات والتي يمكن إعادة استخدامها مما يزيد من كفاء العمل وتقليل الجهد المطلوب.

④**التوثيق الأفضل (Better Documentation) :** وباستخدام أدوات هندسة البرمجيات المساعدة هناك العديد من كميات التوثيق التي يتم توليدها لتمثل

ملاحظات على كيفية تطوير النظام وعمل صيانة له.

⑤البرمجة في أيدي غير المبرمجين :

(Programming in the Hands of Non-programmers) :

مع التطور السريع والاتجاه نحو تكنولوجيا البرمجة الشيئية ، وقواعد بيانات خادم العميل يمكن أن تتم عملية البرمجة بأناس ليس لديهم خلفية كاملة عن البرمجة، حيث يصبح المهم هو فهم الهدف الأساسي من البرنامج والقدرة على تحليل مكونات البرنامج وتفاصيله والتي تستخدم في توليده عن طريق هذه الأدوات (أدوات التطوير منخفضة المستوى).

⑥الفوائد الغير ملموسة (Intangible Benefits) : تفيد أدوات التطوير في مشاركة المستخدم والتي تساهم في قبوله الجيد للنظام الجديد وهذا يساهم إلى حد كبير في تخفيض منحنى التعليم الأولي.

أسئلة تقويم ذاتي

ما هي أدوات هندسة البرمجيات بمساعدة الحاسب الآلي ؟ وما هي أهم مميزاتها وقصور استخدامها؟.

2.5 قصور أدوات هندسة البرمجيات بمساعدة الحاسوب

(CASE Tools Limitations)

يمكن تلخيص أهم القصور الموجودة في أدوات هندسة البرمجيات **بمساعدة الحاسب** في التالي :

①مزج الأدوات (Tool Mix) : من المهم جدا اختيار ملائم لمزيج من الأدوات للحصول على مميزات وتكاليف مخفضة. ولابد من اختيار الأدوات غير المعتمدة على نوعية العتاد وإمكانية مشاركة نتائج أداة تطوير تم استخدامها مع أدوات أخرى لضمان تكاملية الأدوات مع بعضها.

②**التكاليف (Cost)** : الأدوات المساعدة ليست رخيصة الثمن ، وفي الحقيقة فإن شركات التطوير على نطاق ضيق لا تستخدم هذه الوسائل معتقدين بأنها مفيدة فقط في تطوير الأنظمة الكبيرة ، حيث أن تكلفة تزويد مطوري الأنظمة بهذه الوسائل مكلف. ويعتبر (العتاد ، البرمجيات ، الاستشارات) كلها عوامل تدخل في معادلة التكلفة.

③**منحنى التَعلُّم (Learning Curve)** : وفي معظم الحالات فإن إنتاجية المبرمج تصل إلى أدنى مرحلة لها في المرحلة الأولية للتطبيق وذلك لاحتياج المستخدمين للوقت الكافي للتعلم ، بالرغم من أن مستشاري هذه الأدوات يقدمون العديد من الخبرات لمستخدميها حيث يقومون بالتدريب وكذلك وجود العديد من مواقع الخدمات الخاصة بهذه الأدوات على شبكة الإنترنت والتي تساعد في تعجيل منحنى التعليم لهذه الأدوات.

3.5 التصنيف العام لأدوات هندسة البرمجيات المساعدة

(CASE Tools General Classifications) :

❶ **أدوات التطوير المساعدة عالية المستوى (Upper Case Tools) :**

وهي تمثل أدوات أنشطة العمليات المبكرة للمتطلبات ، التحليل والتصميم. وتساعد المحللون في تخزين ، وتنظيم ، وتحليل نماذج العمل ، وترتيب الأسبقيات التالية :

- إستراتيجيات العمل الحالية والمستقبلية.
- النظم المكملة وإستراتيجيات تطبيقها.
- قواعد البيانات والشبكات المراد تطويرها.
- التطبيقات المراد تطويرها.

🕑 **أدوات التطوير المساعدة منخفضة المستوى (Lower Case Tools) :**

وهي تمثل أدوات دعم الأنشطة التالية مثل البرمجة واكتشاف العلل والاختبارات حيث تساعد المبرمجين في زيادة الإنتاجية والجودة ، و تمتد هذه الأدوات إلى تفاصيل التصميم للمساعدة في توليد التطبيقات من خلال الخدمات التالية :

- المساعدة في سرعة اختبار البرنامج واكتشاف الأخطاء.
- توليد الكود للبرنامج من مواصفات التحليل والتصميم.

- توليد الشاشات وقواعد البيانات.

٧ أدوات هندسة البرمجيات بمساعدة الحاسب لدورة حياة التطوير المتقاطعة :

(Cross Life Cycle Case Tools) :

وهي تشمل وسائل إدارة المشاريع التي تساعد المديرين في (التخطيط ، وضع الجداول الزمنية ، وإعداد التقارير ، وتوزيع الموارد).

والجدول رقم (1) يوضح أهم تصنيفات أدوات هندسة البرمجيات بمساعدة الحاسب وأمثلة عليها.

جدول(1) : تصنيف أدوات هندسة البرمجيات بمساعدة الحاسب وأمثلة عليها

نوع الأداة (Tool Type)	أمثلة على الأداة (Examples)
أدوات التخطيط (Planning Tools)	• أدوات بيرت (PERT Tools) • أدوات التقييم (Estimation Tools) • الجداول الإلكترونية (Spreadsheets)
أدوات التحرير (Editing Tools)	• محررات النصوص (Text Editors) • معالجات الكلمات (Word Processors) • محررات المخططات (Diagram Editors)
أدوات إدارة التغيير (Change Management Tools)	• أدوات تتبع المتطلبات (Requirements Traceability Tools) • نظم تغيير التحكم (Change Control Systems)
أدوات إدارة التكوين (Configuration Management Tools)	• نظم إدارة الإصدار (Version Management Systems) • أدوات بناء النظم (System Building Tools)
أدوات النمذجة الأولية (Prototyping Tools)	• اللغات عالية المستوى (Very High-level Languages) • مولدات واجهات المستخدم (User Interface Generators)
أدوات دعم الطرق	• محررات التصميم (Design Editors)

نوع الأداة (Tool Type)	أمثلة على الأداة (Examples)
(Method Support Tools)	• قواميس البيانات (Data Dictionaries) • مولدات الشفرة (Code Generators)
أدوات معالجة اللغات (Language Processing Tools)	• المترجمات (Compilers) • المفسرات (Interpreters)
أدوات تحليل البرامج (Program Analysis Tools)	• مولدات التداخل (Cross reference Generators) • المحللات الإستاتيكية (Static Analyzers) • المحللات الديناميكية (Dynamic Analyzers)
أدوات الاختبار (Testing Tools)	• مولدات اختبار البيانات (Test Data Generators) • مقارنات الملفات (File Comparators)
أدوات اكتشاف وتصحيح الأخطاء (Debugging Tools)	• نظم التصحيح التفاعلية (Interactive Debugging Systems)
أدوات التوثيق (Documentation Tools)	• برامج شكل الصفحة (Page Layout Programs) • محررات الصور (Image Editors)
أدوات إعادة الهندسة (Re-engineering Tools)	• نظم التداخل (Cross-reference Systems) • نظم برامج إعادة الهيكلة (Program Restructuring Systems)

6.التحديات الرئيسة التي تواجه هندسة البرمجيات

(Key challenges Facing Software Engineering) :

تواجه هندسة البرامج ثلاثة تحديات كبرى في القرن الحادي والعشرين ، وهي :

❶ التحدي المتعلق بالموروث (The Legacy Challenge) :

لقد تطورت غالبية الأنظمة البرمجية الرئيسية المستخدمة حاليا منذ سنوات عديدة خلت ، ورغم ذلك لا تزال تلك الأنظمة تؤدي وظائف حساسة. إن التحـــدي المتعلـــق

بالموروث هو التحدي الذي يعني أساسا بصيانة وتحديث البرامج بطريقة يمكن معها تفادي التكاليف الباهظة والاستمرار في الوقت ذاته في تقديم الخدمات الأساسية في قطاع الأعمال.

- **تحديد عدم التجانس (The Heterogeneity Challenge) :**

تزداد الحاجة وباستمرار إلي أنظمة تعمل كأنظمة موزعة عبر شبكات تشتمل على أنواع مختلفة من الحاسبات الآلية ذات الأنظمة المساندة المختلفة وهذا التحدي يعوق تطوير أساليب إعداد برامج يمكن الاعتماد عليها بدرجة كافية للتعامل مع هذا التحدي.

- **تحدي المخرجات (The Delivery Challenge) :**

تُعد العديد من الأساليب المتبعة في هندسة البرامج مضيعة للوقت. كما أن الزمن الذي تستغرقه يحد وبشكل كبير من فرص تطوير البرامج وضمان جودتها . على الرغممن كل ذلك ، فإنه ينبغي على قطاع الأعمال والتجارة في أيامنا هذه أن يتميز بسرعة الاستجابة والتغيير السريع، كما ينبغي أن تتغير وبنفس الوتيرة برامج المساندة الخاصة بها. إن تحدي المخرجات هذا يرتبط بالزمن اللازم للحصول على مخرجات من أنظمة كبيرة ومعقدة دون التفريط في الجودة والنوعية. وبالطبع فإن هذه المسألة ليست مسألة منفصلة فقد يلزم مثلا إجراء تغيير سريع في نظام قديم العهد من أجل جعل النفاذ إليه من خلال الشبكة أمراً سهلا.

وللتعامل مع هذه التحديات يلزمنا توفر أدوات وأساليب جديدة واستخدام طرق إبداعية من أجل القيام بالبرمجة واستخدام الطرق الحالية في هندسة البرمجيات جنباً إلي جنب مع تلك الطرق الإبداعية.

7. مهنة البرمجة (Programming Career)

مهنة البرمجة من المهن الهامة والمطلوبة في السوق العربية بصفة خاصة والعالمية بصفة عامة ، ولكن بشرط أن يكون المبرمج على كفاءة عالية وقدرة على استخدام معظم الأدوات البرمجية وتوظيفها بصورة مثالية، ومن يريد العمل بمهنة البرمجة يجب أن يكون مستخدماً متمرساً للحاسب الآلي وله خبرة طويلة في التعامل مع

شتى أنواع برمجياته ليس فقط كمستخدم عادي ولكن كشخص قادر علـــى فهـــم كيفيـــة تصميم وإنشاء هذه البرمجيات ، مع القدرة على اكتساب المهـــارات الخاصـــة بـــأدوات البرمجة وتطويرها ، وكذلك يكون له عقل يجيد التعامل مع الأسس الرياضية ، حيث إن مهنة البرمجة لا تعتمد على مجرد أداء المهام فقط وإنما تتطلب فكـــراً خصـــباً وذهنـــاً حاضراً وحباً للإبداع في العمل والمثابرة عليه.

ولتصميم وتطوير البرامج والتطبيقات بطريقة قياسية ، يجـــب أن يتكـــون فريـــق العمل الخاص بذلك من التالي :

- **محللو النظم (System Analysts)** : وهم الأشخاص القائمون علي دراســـة وتحليـــل متطلبات النظام ومدخلاته ومخرجاته ، وكذلك تحديد الموارد اللازمة لتنفيذه ، بالإضافة إلى بيان كيفية التنفيذ وشرح ديناميكية العمل وتنظيم العلاقات المختلفـــة بـــين الكائنـــات الموجودة بالنظام .
- **مصممو النظم (System Designers)** : يأتي دورهم بعد مرحلـــة التحليـــل وتحديـــد الاحتياجات ، حيث يكون النظام بحاجة الآن إلي كيفية التطبيق من حيث الشكل العام وتصـــميم كائنات ونماذج النظام وبنية كل كائن علي حده. وفي هذه المرحلة يتم تصميم نمـــاذج وأشـــكال الشاشات ومواضعها وطرق عرضها وربطها مع بعضها البعض.
- **المبرمجون (المطورون)** (**Programmers (Developers)**) : ويـــأتي دورهـــم بعـــد مرحلتي التحليل والتصميم حيث يتم التنفيذ الفعلي للنماذج والشاشات المصـــممة وكتابـــة الشفرات **(Source Code)** المسئولة بدورها عن تشغيل النظام .

فمثلا إذا كنا بصدد إنشاء نظام لإدارة شركة ما من الناحية الماليـــة والتجاريـــة ، فسيقوم المحللون بدراسة الدورة المستندية لهذه الشركة ، وكيفية تعاملها مـــع الشـــركات الأخرى ، وديناميكية العمل من حيث المستندات المستخدمة في دورات العمل المختلفـــة ...الخ ، وكيفية تدفق البيانات من مرحلة إلي الأخرى ، وبالتالي تحليل النظام ككل بشكل متكامل ثم. بعد ذلك يأتي دور المصممين حيث يتم تصميم نمـــاذج وأشـــكال الشاشـــات ومواضعها وطريقة عرضها وربطها ببعض والتي سيبرمجها المبرمجون ، وبعـــد ذلـــك

يأتي دور المبرمجين حيث يتم التنفيذ الفعلي لما تم تصميمه سابقا حيث يتم كتابة الشفرات اللازمة لإنشاء كل النماذج وربطها بعضها ببعض.

ومن خلال ما سبق ، يتضح أن المبرمج هو الشخص القائم على كتابة الشفرات اللازمة لبث روح الحياة في النظام وجعله وحدة واحدة مترابطة يؤدي في النهاية – عند تشغيله من قبل المستخدم – جميع المهام الذي صمم من أجلها ، وبالتالي فالمبرمج هو حلقة الوصل بين الحاسب الآلي والمستخدم ، فكلاهما لا يعرف لغة الآخر ، ولكن المبرمج يعرف لغة الاثنين.

والمبرمج ليس من تعلم لغة برمجية فحسب ، بل المبرمج هو من يعرف فن البرمجة ، أي كيف يضع الإستراتيجية المناسبة في المكان المناسب ، وهذه بحد ذاتها موهبة ربانية كالرسم والنحت ، أما ما تبقى من العمل البرمجي فلا يتعدى تطويع الذخائر والأدوات البرمجية وتشكيل الفكرة في قالب ذي طابع فني برمجي لإنتاج المنتج البرمجي.

ولكي يبدأ أي شخص بامتهان مهنة البرمجة كوظيفة يجب عليه أولاً التعرف على أنواع لغات البرمجة من حيث نقاط القوة والضعف في كل منها ، وكذلك التطبيقات الخاصة بكل منها ، وتعلُّم مبادئ البرمجة ومفاهيمها الأساسية والمشتركة بين جميع لغات البرمجة ، وهو الهدف الأساسي من هذا الكتاب الذي بين يديك ، فإذا كنت تريد أن تبدأ في عالم البرمجة فعليك بدراسة هذا الكتاب وبالتسلسل المذكور به.

أسئلة تقويم ذاتي

1.ما هي أهم التحديات الرئيسية التي تواجه هندسة البرمجيات؟ وكيف يمكن مواجهتها من وجه نظرك؟

2. الجدول التالي يحتوي العمود الأول منه على الفرق الأساسية لتطوير البرمجيات والعمود الثاني على وظائف هذه الفرق، أعد بناء الجدول بحيث يكون هناك توافق بين الفرق ووظائفها

فرق تطوير البرمجيات	وظائفها
محللو النظم	• يقوم بتصميم نماذج وأشكال الشاشات ومواضعها وطرق عرضها وربطها مع بعضها البعض.
مصممو النظم	• يقوم بتحليل متطلبات النظام ومدخلاته ومخرجاته وكذلك تحديد الموارد اللازمة لتنفيذه.
المبرمجون	• يقوم بكتابة الشفرات المسئولة بدورها عن تشغيل النظام.

8. المسئولية المهنية والأخلاقية لمهندسي البرمجيات

(Professional and Ethical Responsibility for Software Engineers) :

مثلهم في ذلك مثل غيرهم من المهندسين ، فإنه يتوجب على مهندسي البرمجيات القبول بحقيقة أن مهامهم تتضمن مسئوليات أكبر من مجرد استخدام المهارات الفنية حيث أن عملهم يتم ضمن إطار قانوني واجتماعي معين، ومن الواضح أن هندسة البرمجيات مقيدة بمجوعة من القوانين المحلية والوطنية والدولية، وتبعاً لذلك فإن عليهم

أن يتصرفوا بطريقة مسئوله من الناحتين الأخلاقية والمعنوية إذا كان لهم أن يحوزوا على الاحترام كمحترفين في مجالات عملهم، ومن البديهي كذلك أن يتمسك هؤلاء المهندسون بالقواعد والمعايير المتعارف عليها في مجال الشرف والنزاهة والأمانة والاستقامة. كما ينبغي عليهم عدم استخدام مهاراتهم وقدراتهم بطريقة غير شريفة أو تسيء إلي سمعة المهنة، ومع ذلك فإن هناك مجالات لا تكون فيها معايير السلوك المقبول ملزمة بموجب القانون بل ترتبط بمفاهيم فضفاضة للمسئولية المهنية، وهذه بعض منها :

- **السرية (Confidentiality)** : يجب على المهندسين في العادة احترام خصوصية وسرية المعلومات الخاصة بعملائهم وأرباب العمل بصرف النظر عما إذا تم توقيع اتفاقية خاصة بالسرية أم لا.
- **الاختصاص (Competence) :** يتوجب علي المهندسين الامتناع عن إعطاء معلومات خاطئة عن مستوي تأهيلهم واختصاصهم ، كما أن عليهم عدم قبول أي عمل خارج عن نطاق اختصاصهم وهم يعلمون ذلك.
- **حقوق الملكية الفكرية (Intellectual Property Rights) :** يجب أن يكون المهندسين مطلعين على القوانين المحلية التي تحكم استخدام الممتلكات الفكرية مثل براءات الاختراع وحقوق النشر والطبع والتأليف وغيرها، كما يجب عليهم أن يحرصوا على التأكد من حماية الممتلكات الفكرية لأرباب عملهم وعملائهم.
- **إساءة استخدام أجهزة الحاسب الآلي (Computer Misuse) :** ينبغي على مهندسي البرمجيات عدم استخدام مهاراتهم الفنية في الإساءة إلي أجهزة الحاسب الآلي التي تعود للآخرين وإساءة الاستخدام تتراوح بين أمور تافهة نسبياً (مثلً ممارسة الألعاب على جهاز صاحب العمل) وأمور خطيرة جداً (كنشر الفيروسات مثلاً).

وفى هذا الخصوص تلعب الجمعيات والمؤسسات المهنية دورا هاما ، فالمنظمات مثل : "اتحاد أصحاب الآلات الحاسبة" ، و"معهد المهندسين الكهربائيين والإلكترونيين" ، و"جمعية الحاسبات البريطانية" تقوم بنشر لوائح السلوك المهني أو قواعد أخلاق المهنة. كما يتعهد أعضاء هذه المنظمات بالتقييد بتلك اللوائح والقواعد عند

التوقيع على العضوية.

وينبغي على مهندسي البرمجيات الالتزام بجعل مهنة تحليل وتصـــميم وتطـــوير واختبار وصيانة البرامج مهنة محترمة وذات فائدة وجدوى، ووفـــق التـــزامهم بصـــحة وسلامة ورفاهية الجمهور فإن على مهندسي البرامج الالتزام بالمبادئ الثمانية التالية :

❶ **الجمهور (Public) :** يجب على مهندسي البرمجيات العمل بما يتوافق مع المصلحة العامة.

❷ **العميل وصاحب العمل (Client And Employer) :** يجب على مهندسي البرمجيات العمل بطريقة تخدم مصالح عملائهم وأرباب الأعمال وتكون متوافقة مع المصلحة العامة.

❸ **المنتج (Product) :** يجب على مهندسي البرمجيات التأكد من أن منتجاتهم والتعديلات المتعلقة بها تفي بأعلى المستويات المهنية الممكنة.

❹ **النزاهة (Judgment)** : يجب على مهندسي البرامج المحافظة على التكامل والاستقلالية في أرائهم المهنية.

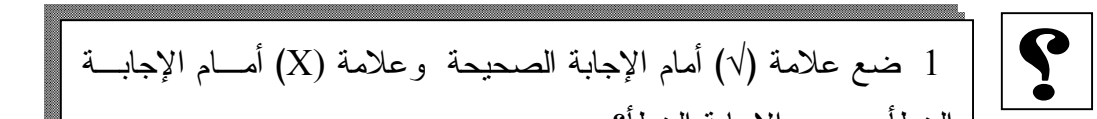
❺ **الإدارة (Management)** : يجب على المدراء والقياديين في مجال هندسة البرمجيات المساهمة والتعهد بالالتزام بالمنهج الأخلاقي في إدارة عملية تطوير وصيانة البرمجيات.

❻ **المهنة (Profession)** : يجب على مهندسين البرمجيات تحسين سمعة وصورة المهنة وبما يتوافق مع المصلحة العامة.

❼ **الزملاء (Colleagues)** : يجب على مهندسي البرامج أن يكونوا منصفين وعادلين مع زملائهم وان يقدموا الدعم والمساندة لهم.

❽ **الذات (Self)** : يجب على مهندسي البرامج المشاركة في برامج التعليم الدائم والمتعلق بممارسة مهنتهم كما أن عليهم تعزيز الأسلوب الأخلاقي في ممارسة هذه المهنة.

أسئلة تقويم ذاتي



1 ضع علامة (√) أمام الإجابة الصحيحة وعلامة (X) أمام الإجابة الخطأ. صحح الإجابة الخطأ؟ :

- هندسة النظم جزء من هندسة البرمجيات.
- مصطلح هندسة البرمجيات يحتوي على شقين أحدها "هندسي" تطبق فيه النظريات والأساليب للحصول على الحلول المثالية للمشاكل والأخر "برمجي" يهتم بجميع مظاهر إنتاج البرمجيات.
- هندسة البرمجيات لا تهتم فقط بالعملية الفنية لتطوير البرمجيات ولكن تهتم أيضا بالنشاطات الأخرى مثل إدارة مشروع البرمجيات.
- علوم الحاسبات تركز على أنظمة الحاسبات ولغات البرمجة لدراستها وتطويرها بينما هندسة البرمجيات تهتم بالمشاكل العملية لإنتاج البرمجيات.
- تحديد المتطلبات وقيود تشغيلها وتطوير النظام البرمجي والتحقق من أداء النظام البرمجي عبارة عن الأنشطة الأربعة الأساسية في جميع عمليات البرمجيات.
- أدوات هندسة البرمجيات بالاستعانة بالحاسب عبارة عن أنظمة برمجية مصممة لإسناد الأنشطة الروتينية لعمليات إنتاج البرامج.

2. رتب (ترتيبا تصاعديا) المهام التالية التي يجب أن تتبع في تطوير المشاريع البرمجية :

إدارة المشاريع ، صيانة البرمجية ، تصميم البرمجية ، اختبار تكاملية البرمجية ، تحديد المتطلبات، توثيق البرمجية، تقدير الموارد، الترميز(عملية البرمجة)، التثبيت وتوزيع البرمجية، التدريب، تحليل المشكلة.

اسئلة تقويم ذاتي

اختر المصطلح المناسب والصحيح لكل من التعريفات التالية :

- تسليم المنتج البرمجي في الوقت المحدد.
- إمكانية التشغيل والتعامل مع أنظمة أخرى عادية أو شبكية.
- يمكن إعادة استخدامها مع مشاريع أخرى لأنها مكتوبة وفقا لمعايير.
- يمكن اختبارها بسهولة.
- يمكن نقلها إلى أي نظام تشغيل أو حاسب بدون إعادة كتابة أي أجزاء جديدة.
- تعني توفر ربط بيني مناسب للمستخدم ومادة وثائقية كافية.
- يمكن صيانته إذا حدثت أي مشاكل أثناء التشغيل.
- تعني تنفيذ البرمجية في الوقت المحدد من خلال الموارد المتاحة.
- تعني عدم تذبذب أداء البرمجية من وقت لآخر.
- يتصرف النظام بشكل يتوافق مع المتطلبات الوظيفية.

الخلاصة

اشتملت هذه الوحدة على تعريف هندسة البرمجيات والتدقيق في مصطلح "هندسة البرمجيات" بشقيه الهندسي الذي يطبق فيه المهندسون النظريات والأساليب والأدوات الملائمة ، الشق البرمجي والذي يهتم بجميع مظاهر إنتاج البرمجيات.

كما استلمت على المهام النموذجية لهندسة البرمجيات وهي تحليل المشكلة وتحديد المتطلبات وتصميم البرمجية والترميز اختبار تكاملية الشفرة البرمجية والتثبيت وتوزيع البرمجية وتوثيق البرمجية وصيانة البرمجية. مبادئ هندسة البرمجيات وهي الدقة والفورية ، فصل الاهتمامات ، التجزئة ، توقع التغيير ، العمومية ، إعادة الاستخدام.

أنواع البرمجيات وهي برمجيات النظام والبرمجيات التطبيقية وبرمجيات الألعاب وبرمجيات نظم المعلومات وبرمجيات نظم الوقت الحقيقي والبرمجيات المدمجة بالأجهزة والبرمجيات المكتبية والبرمجيات العلمية والهندسية وبرمجيات الذكاء الصناعي.

أدوات هندسة البرمجيات بمساعدة الحاسب الآلي والتي تمثل مجموعة من الضوابط والآليات التي تساعد في تحليل ، وتصميم ، ونمذجة وترميز ، واختبار ، وتوثيق البرمجيات.

للتحديات الرئيسة التي تواجه هندسة البرمجيات والمتمثلة في التحدي المتعلق بالموروث – تحديد عدم التجانس – تحديد المخرجات.

مهنة البرمجة كما تناولت أخيراً التصميم وتطوير البرامج والتطبيقات بطريقة قياسية ، يجب أن يتكون فريق العمل الخاص بذلك من : محللي النظم ، مصممي النظم ، المبرمجين.

المسئولية المهنية والأخلاقية لمهندسي البرمجيات وهي لوائح السلوك المهني وهي السرية والاختصاص وحقوق الملكية الفكرية وإساءة استخدام أجهزة الحاسب الآلي. كم أن هناك مبادئ يجب أن يلتزم بها مهندس البرامج وهي الجمهور والعميل والمنتج والنزاهة والإدارة، المهنة والزملاء والذات.

أرجو أن تساهم هذه الخلاصة في مساعدتك على استذكار ما ورد في هذه الوحدة

إجابة التدريبات

تدريب (1)

بصورة أساسية ، تهتم **علوم الحاسبات الآلية** بالنظريات والطرق التي تؤكد دراسة أنظمة الحاسبات والأدوات البرمجية ، بمعنى أنها تركز على أنظمة الحاسبات ولغات البرمجة لدراستها وتطويرها في ذاتها ، أي أنها تُعد أداة نستخدمها عند تصميم وتطوير حل لمشكلة ما ، وذلك مثل علم الكيمياء حيث يهتم الكيميائي بالمواد الكيميائية ذاتها من خلال دراسة تركيباتها ، وتفاعلاتها ، والنظريات التي تحكم سلوكها. في حين أن **هندسة البرمجيات** تهتم بالمشاكل العملية لإنتاج البرمجيات ، أي أن مهندس البرمجيات يعتبر أن الحاسبات الآلية أداة لحل المشاكل ، وذلك مثل المهندس الكيميائي الذي يعتبر الكيمياء أداة لإيجاد الحلول لمشاكل عامة في الصناعات الكيميائية وغيرها. ومعرفة بعض علوم الكمبيوتر ضرورية وأساسية لمهندسي البرمجيات وبنفس الطريقة فإن بعض المعرفة للفيزياء ضرورية لمهندسي الكهرباء. مثالياً ، يجب أن تعزز جميع عمليات هندسة البرمجيات بنظريات علوم الحاسبات الآلية ولكن في الحقيقة ليست هذه هي القضية أو الحالة ، بل يجب أن يستخدم مهندسو البرمجيات أحياناً أساليب خاصة لتطوير البرمجيات ، حيث إن نظريات علوم الحاسبات الآلية لا يمكن تطبيقها دائماً للمشاكل الحقيقة والمعقدة التي تتطلب حلول برمجية خاصة.

تدريب (2)

تهتم **هندسة النظم** بجميع العمليات المتعلقة بتطوير الأنظمة المعتمدة على الحاسبات الآلية **(Computer-Based Systems)** بما في ذلك الأجزاء الصلبة للحاسبات الآلية **(Hardware)** والبرمجيات **(Software)** والعمليات الهندسية المتعلقة بعمليات التطوير المختلفة والتي تتضمن هندسة البرمجيات ، أي أن هندسة البرمجيات جزء من هندسة النظم. ويختص مهندس النظم بتعريف وتحديد هندسة النظام الكاملة ومن ثم تكامل الأجزاء المختلفة لخلق وابتكار النظام المكتمل.

تدريب (3)

عمليات البرمجيات هي مجموعة من الأنشطة المرتبة (الطبقية) المترابطة **(Consistency Activities)** والقيود **(Constraints)** والموارد **(Resources)** تهدف إلى توصيف **(Specifying)** وتصميم **(Designing)** وتنفيذ **(Implementing)** واختبار **(Testing)** وصيانة وترقية **(Maintenance and Evolution)** منتج برمجي يفي بجميع متطلبات العميل. وغالباً ما يقوم بهذه الأنشطة مهندسو البرمجيات. وتوجد أربعة أنشطة أساسية في جميع عمليات البرمجيات ، وهي:

- **تحديد المتطلبات وقيود تشغليها وهي** ما يجب أن يقوم به النظام البرمجي والقيود المفروضة أثناء عملية التطوير.
- **تصميم وتنفيذ (تطوير) النظام البرمجي.**
- **التحقق من أداء النظام البرمجي وهو يعني** التأكد من أن النظام البرمجي يفي بجميع متطلبات العميل ومقاييس الجودة المطلوبة.
- **الصيانة والارتقاء وهو** صيانة وتغيير بعض من أجزاء النظام البرمجي تجاوباً مع المتغيرات في متطلبات العميل.

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس ،،

بعد أن فرغت من أهم المفاهيم الأساسية في هندسة البرمجيات في الوحدة الأولى ينتقل بك المقرر إلى الوحدة التالية التي سنتابع فيها دورة حياة (عمليات) البرمجيات من مرحلة تجميع المتطلبات إلى مرحلة التطوير ثم مرحلة الاختبار وتشخيص الأخطاء وأخيراً مرحلة الصيانة والارتقاء .

نرجو أن تكون وحدة مفيدة لك وأن تساهم معنا في نقدها وتقويمها.

مسرد المصطلحات

هندسة البرمجيات Software Engineering :

علم يهتم ببناء الأنظمة البرمجية الكبيرة والمعقدة بواسطة فريق من مهندسي البرمجيات بإتباع طرق ومبادئ هندسية منظمة وصحيحة واستخدام وسائل وأدوات وتقنيات تجعل من تصميم وتطوير وبناء واختبار هذه النظم عملية منظمة يمكن تكرارها في حدود الإمكانيات المتاحة مادياً وبشرياً وزمنياً ، وذلك من خلال دراسة دورة حياة النظام مع إعطاء أشكالاً متعددة لعمليات تصميمه وتطويره وبناؤه واختباره بحيث يكون المنتج البرمجي النهائي على درجة عالية من الجودة والموثوقية وقليل التكاليف بقدر الإمكان ويتم تسليمه للعميل في الوقت المناسب ويعمل بكفاءة على أجهزة الحاسب الآلي مختلفة.

علوم الحاسبات الآلية Computer Sciences:

هي العلوم التي تهتم بالنظريات والطرق التي تؤكد على دراسة أنظمة الحاسبات والأدوات البرمجية ، بمعنى أنها تركز على أنظمة الحاسبات ولغات البرمجة لدراستها وتطويرها في ذاتها ، أي أنها تُعد أداة نستخدمها عند تصميم وتطوير حل لمشكلة ما ، وذلك مثل علم الكيمياء حيث يهتم الكيميائي بالمواد الكيميائية ذاتها من خلال دراسة تركيباتها ، وتفاعلاتها ، والنظريات التي تحكم سلوكها.

هندسة النظم System Engineering :

تهتم هندسة النظم بجميع العمليات المتعلقة بتطوير الأنظمة المعتمدة على الحاسبات الآلية (Computer-Based Systems) بما في ذلك الأجزاء الصلبة للحاسبات الآلية (Hardware) والبرمجيات (Software) والعمليات الهندسية المتعلقة بعمليات التطوير المختلفة والتي تتضمن هندسة البرمجيات ، أي أن هندسة البرمجيات جزء من هندسة النظم.

عمليات البرمجيات Software Processes:

هي مجموعة من الأنشطة المرتبة (الطبقية) المترابطة (Consistency Activities) والقيود (Constraints) والموارد (Resources) تهدف إلى توصيف (Specifying) وتصميم (Designing) وتنفيذ (Implementing) واختبار (Testing) وصيانة وترقية (Maintenance and Evolution) منتج برمجي يفي بجميع متطلبات العميل.

جودة البرمجية Quality Assurance:

مصطلح يتعلق بعملية تطوير المنتج البرمجي والطريقة التي تم تطويره بها بحيث تكون وفقا لمعايير مقاييس الجودة والتي يتم وضعها من خلال مجموعة مراقبة الجودة من فريق التطوير بعد تحديد متطلبات المشروع البرمجي مباشرة وقبل البدء في التنفيذ الفعلي للمشروع.

التجزئة Modularity:

تقسيم النظام البرمجي المعقد إلى أقسام أبسط تسمى كتل.

التجريد Abstraction :

تحديد المظاهر المهمة وتجاهل تفاصيلها أثناء دراسة مسالة معينة (فصل الشيء المهم عن الشيء غير المهم في المسألة) ، وهي نظرة نسبية مرتبطة بالهدف الذي نعمل عليه.

توقع التغيير Anticipation of Change :

إحصاء الأماكن التي يمكن أن يتم تعديل عليها في البرمجيات.

إعادة الاستخدام Reuse :

يشير هذا المصطلح إلى تطوير برمجية معتمده على استخدام أجزاء جاهزة تم إعدادها وفقا لمعايير الجودة حيث تم اختبارها من قبل مما يزيد من إنتاجية المشروع وتوفير المزيد من الوقت. ويمكن النظر لهذا المصطلح – أيضاً – على أنه استخدام ما هو متاح فعليا لتحقيق ما هو مطلوب.

محللو النظم System Analysts:

وهم الأشخاص القائمون علي دراسة وتحليل متطلبات النظام ومدخلاته ومخرجاته ، وكذلك تحديد الموارد اللازمة لتنفيذه ، بالإضافة إلى بيان كيفية التنفيذ وشرح ديناميكية العمل وتنظيم العلاقات المختلفة بين الكائنات الموجودة بالنظام .

المصطلح بالإنجليزية	معناه بالعربية
Abstraction	التجريد
Anticipation of Change	توقع التغيير
Application Software	البرمجيات التطبيقية
Artificial Intelligence Software	برمجيات الذكاء الاصطناعي
Attributes of Software	الخصائص المميزة للبرمجيات
Better Documentation	التوثيق الأفضل
CASE	هندسة البرمجيات بمساعدة الحاسب
Change Control Systems	نظم تغيير التحكم
Change Management Tools	أدوات إدارة التغيير
Client And Employer	العميل وصاحب العمل
Code Generators	مولدات الشفرة
Competence	الاختصاص
Compilers	المترجمات
Computer Misuse	إساءة استخدام أجهزة الحاسب الآلي
Computer Sciences	علوم الحاسبات الآلية
Confidentiality	السرية
Configuration Management Tools	أدوات إدارة التكوين
Consistency Activities	الأنشطة المرتبة (الطبقية) المترابطة
Correctness	الصحة
Cross Life Cycle Case Tools	أدوات دورة حياة التطوير المساعدة المتقاطعة
Cross reference Generators	مولدات التداخل
Cross-reference Systems	نظم التداخل
Data Dictionaries	قواميس البيانات
Debugging Tools	أدوات اكتشاف وتصحيح الأخطاء
Dependability	الموثوقيه
Design Editors	محررات التصميم

المصطلح بالإنجليزية	معناه بالعربية
Developers	المطورون
Diagram Editors	محررات المخططات
Documentation Tools	أدوات التوثيق
Dynamic Analyzers	المحللات الديناميكية
Editing Tools	أدوات التحرير
Efficiency	الكفاءة
Embedded Software	البرمجيات المدمجة بالأجهزة
Engineering and Scientific Software	البرمجيات العلمية والهندسية
Estimation Tools	أدوات التقويم
File Comparators	مقارنات الملفات
Games Software	برمجيات الألعاب
Generality	العمومية
Image Editors	محررات الصور
Implementing	تنفيذ
Increased Accuracy	الدقة المتزايدة
Increased Speed	السرعة الفائقة
Information Systems Software	برمجيات نظم المعلومات
Intangible Benefits	الفوائد غير الملموسة
Intellectual Property Rights	حقوق الملكية الفكرية
Interactive Debugging Systems	نظم التصحيح التفاعلية
Interpretability	إمكانية تشغيلها مع برمجيات أخرى
Interpreters	المفسرات
Judgment	المحاكمة العقلية
Key challenges facing Software Engineering	التحديات الرئيسية التي تواجه هندسة البرمجيات
Language Processing Tools	أدوات معالجة اللغات

المصطلح بالإنجليزية	معناه بالعربية
Learning Curve	منحنى التعليم
Lower Case Tools	أدوات التطوير المساعدة منخفضة المستوى
Maintainability and Evolubility	سهولة الصيانة وقابلية التطوير
Maintenance and Evolution	صيانة وترقية
Management	الإدارة
Method Support Tools	أدوات دعم الطرق
Modularity	التجزئة
Office Software	البرمجيات المكتبية
Page Layout Programs	برامج شكل الصفحة
PERT Tools	أدوات بيرت
Portability	إمكانية الحمل أو النقل
Product	المنتج
Profession	المهنة
Professional and Ethical Responsibility for Software Engineers	المسئولية المهنية والأخلاقية لمهندسي مسئوليات مهندسي البرمجيات
Program Analysis Tools	أدوات تحليل البرامج
Program Restructuring Systems	نظم برامج إعادة الهيكلة
Programmers	المبرمجون
Programming Career	مهنة البرمجة
Programming in the Hands of Non-Programmers	البرمجة في أيدي غير المبرمجين
Prototyping Tools	أدوات النمذجة الأولية
Public	الجمهور
Quality Assurance	جودة الأداء
Real – Time Systems Software	برمجيات نظم الوقت الحقيقي
Reduced Lifetime Maintenance	تخفيض الوقت اللازم للصيانة

المصطلح بالإنجليزية	معناه بالعربية
Re-engineering Programs	برمجيات إعادة الهندسة
Re-engineering Tools	أدوات إعادة الهندسة
Reliability	الاعتمادية
Requirements Traceability Tools	أدوات تتبع المتطلبات
Requirements Traceability Tools	أدوات تتبع المتطلبات
Resources	الموارد
Reusability	إعادة الاستخدام
Reuse	إعادة الاستخدام
Rigor And Formality	الدقة والصورية
Separation of Concern	فصل الاهتمامات
Software Engineering	هندسة البرمجيات
Software Engineering Tasks	المهام النموذجية لهندسة البرمجيات
Software Industry (Development)	صناعة البرمجيات
Software Life Cycle	أطوار دورة حياة تطوير البرمجيات
Software Processes	عمليات البرمجيات
Specifying	توصيف
Static Analyzers	المحللات الإستاتيكية
System Analysts	محللو النظم
System Building Tools	أدوات بناء النظم
System Designers	مصممو النظم
System Engineering	هندسة النظم
System Software	برمجيات النظام
Test Data Generators	مولدات اختبار البيانات
Testability	إمكانية اختبارها
Testing	اختبار
Testing Tools	أدوات الاختبار

المصطلح بالإنجليزية	معناه بالعربية
The Delivery Challenge	تحدي المخرجات
The Heterogeneity Challenge	تحديد عدم التجانس
The Heterogeneity Challenge	تحديد عدم التجانس
The Legacy Challenge	التحدي المتعلق بالموروث
Timeliness	ضبط وقت
Tool Mix	مزج الأدوات
Upper Case Tools	أدوات التطوير المساعدة عالية المستوى
Usability or User Friendliness	قابلية الاستخدام أو سهولة التعامل
User Interface Generators	مولدات واجهات المستخدم
Version Management Systems	نظم إدارة الإصدار

المراجع

أولاً : المراجع العربية :

[1] روجر بريسمان ، "هندسة البرمجيات" ، الدار العربية للعلوم ، مركز التعريب والبرمجة ، الطبعة الأولى ، 1425هـ - 2004م.

[2] مهندس إياد هلالي ، "هندسة البرمجيات" ، المركز الألماني السوري لأعمال الانترنت **(gsibc.net)** ، منشور على الموقع :

http://www.w3.org

[3] أسماء المنقوش ، "دورة هندسة البرمجيات" ، منشور على الموقع :

http://www.c4arab.com/

المرجع الأساسي لهذه الدورة هو :

Shari Pfleeger, "Software Engineering - Theory and Practice", 2nd Edition

[4] مهندس عبد الحميد بسيوني ، "أساسيات هندسة البرمجيات" ، دار الكتب العلمية للنشر والتوزيع ، القاهرة ، 2005 م.

ثانياً : المراجع الإنجليزية :

[5] Ian Somerville , "Software Engineering", Addison Wesley, 2001.

[6] Ronald J. Leach,, "Introduction to Software Engineering", CRC Press, 1999.

[7] Douglas Bell , "Software Engineering A Programming Approach", 3rd Edition, Addison Wesley.

[8] **Shari Pfleeger, "Software Engineering - Theory and Practice", 2nd Edition.**

ثالثاً : مواقع على شبكة الإنترنت تم الاستفادة منها :

[9] www.qucis.queensu.ca/Software-Engineering/

[10] www.c4arab.com/

[11] http://forum.amrkhaled.net/

[12] http://ar.wikipedia.org/wiki

[13] http://www.uop.edu.jo/Arabic/Faculties/

[14] http://cs.wwc.edu/

[15] http://www.pcwebopedia.com/

الوحدة الثانية

دورة حياة (عمليات) البرمجيات

محتويات الوحدة

مقدمة

تمهيد

عزيزي الدارس ،،

أهلاً بك في هذه الوحدة من مقرر " هندسة البرمجيات(1)" هذه الوحدة تتناول دورة حياة (عمليات) البرمجيات ، وهي دراسة تفصيلية للمراحل المختلفة لدورة حياة البرمجيات.

تركز المرحلة الأولى من مراحل حياة (عمليات)البرمجيات على دراسة متطلبات حل المشكلة، وتنتهي هذه المرحلة بكتابة "وثيقة توصيف المتطلبات" التي تشمل المواصفات الفنية للمشروع البرمجي بناءاً على متطلبات العميل.

تنقسم المرحلة الثانية إلى مرحلتين : مرحلة التصميم حيث تعد "وثيقة توصيف المتطلبات" من معطيات هذه المرحلة و مرحلة التنفيذ حيث تعد أيضاً "وثيقة توصيف المتطلبات" من معطيات هذه المرحلة وتتم هذه المرحلة على خطوتين ؛ الأولى عملية البرمجة والثانية عملية التكامل.

المرحلة الثالثة من مرحلة الاختبار وتشخيص الأخطاء وهي مرحلة مستمرة خلال جميع مراحل تطوير المشروع البرمجي، وتشمل هذه المرحلة خطوتين : الأولى خطوة التحقق ، والثانية خطوة المصادقة.

المرحلة الرابعة وهي مرحلة الصيانة والارتقاء حيث تبدأ من هذه المرحلة أهم مراحل عمر المنتج وهي مرحلة الصيانة.

وهي عبارة عن مجموعة من الأنشطة المستقلة عن أي نشاط برمجي هيكلي وتجري طوال عملية بناء المنتج البرمجي.

تجد عزيزي الدارس فى ثنايا الوحدة بعض التدريبات التي يساعدك تنفيذها على فهم محتوى المادة هذا بالإضافة إلى أسئلة التقويم الذاتي التي تهدف إلى ترسيخ الفهم وتعزيزه لديك و تعميقه .

أهداف الوحدة

في نهاية هذه الوحدة ينبغي أن يكون قادراً على أن :

- تعدد عمليات البرمجيات.
- تصف هيكلاً عاماً لعمليات البرمجيات.
- تشرح دورة حياة البرمجيات.
- تدون أهم مراحل تطوير البرمجيات وأهمية كل منها في عملية التطوير.
- تحدد مدخلات ومخرجات مرحلة تجميع المتطلبات.
- تصف بعض الطرق الممكن استخدامها في مرحلة تجميع المتطلبات.
- تحدد مدخلات ومخرجات مرحلة التصميم.
- تصف بعض الطرق الممكن استخدامها في عملية تصميم البرمجيات.
- تحدد مدخلات ومخرجات مرحلة التصميم (البرمجة والتكامل).
- تشرح الفرق بين عمليتي المصادقة على والتحقق من صلاحية البرمجيات .
- تسرد أهم الأخطاء التي يمكن اكتشافها أثناء عملية اختبار البرمجيات.
- تعبر عن أهمية صيانة البرمجيات ، ويسرد أهم الأعمال التي قد تشملها.

توطئة

سبق وأن عَّرفنا في الوحدة الأولى عمليات البرمجيات بأنها : هي مجموعة من الأنشطة المرتبة (الطبقية) المترابطة **(Consistency Activities)** والقيود **(Constraints)** والموارد **(Resources)** **التى** تهدف إلى توصيف **(Specifying)** وتصميم **(Designing)** وتنفيذ **(Implementing)** واختبار **(Testing)** وصيانة وترقية **(Maintenance and Evolution)** منتج برمجي يفي بجميع متطلبات العميل، وغالباً ما يقوم بهذه الأنشطة مهندسوا البرمجيات. ويمكن وصف عمليات البرمجيات بوضع هيكل عام لها **(Common Process Framework)** ، وذلك من خلال وضع عدد محدد من نشاطات الهيكل **(Framework Activities)** والتي يمكن تطبيقها على جميع المشاريع البرمجية بغض النظر عن حجمها أو تعقيدها ، وعدد من مجموعات المهام **(Task Set)** اللازمة لكل نشاط هيكلي ، هذا بالإضافة إلى مجموعة من نشاطات هندسة البرمجيات والتي يطلق عليها " نشاطات المظلة "، وهي عبارة عن مجموعة من الأنشطة مستقلة عن أي نشاط برمجي هيكلي وتُجرى طوال عملية بناء المنتج البرمجي (مثل : متابعة المشروع البرمجي ومراقبته ، ضمان جودة المنتج البرمجي ، إدارة تكوين البرنامج ، إدارة المخاطر ، وخلافه) ، كما سيتم توضيحه فيما بعد.

فمثلاً ، يُعد الاتصال بالعميل نشاط هيكلي يحتاج إلى مجموعة من المهام منها على سبيل المثال : إعداد وثيقة عمل وجدول أعمال للاجتماع الرسمي مع العميل ، وإجراء أبحاث لتحديد الحلول المقترحة والطرق الموجودة لمعالجة الموضوع و تجميع متطلبات العميل ، مراجعة متطلبات العميل مع جميع المعنيين بالأمر ، وخلافه من المهام.

وتشير تلك العمليات **(Software Process)** ــ في الوقت نفسه ــ إلى مراحل تطوير البرمجيات. وبمفهوم علم هندسة البرمجيات **(Software Engineering)** يُطلق على تلك العمليات – أيضاً – اصطلاح عام وهو دورة حياة البرمجيات **(Software**

(Life Cycle) . وعلم هندسة البرمجيات هو علم يهدف إلى إنتاج برمجيات خالية من الأخطاء وذات جودة عالية في وقت محددة وبميزانية محددة ، وبطريقة اقتصادية بحيث يفي بجميع متطلبات الجهة المستفيدة.

وبغض النظر عن مجال التطبيق أو حجمه أو درجة تعقيده ، وبدون الدخول في تفصيلات النماذج **(Models)** التي تصف عمليات المراحل مثل : **Waterfall Model** و **Spiral Model** ، والتي سوف نتناولها في الوحدة الثالثة من هذا الكتاب ، يمكن تقسيم مراحل تطوير البرمجيات إلى المراحل الرئيسية التالية :

1. مرحلة تحليل المشكلة **(Problem Analysis Phase)**.
2. مرحلة التطوير **(Development Phase)** وتشمل مرحلتي التصميم **(Design Phase)** والتنفيذ **(Implementation Phase)**.
3. مرحلة الاختبار وتشخيص الأخطاء **(Testing and Debugging Phase)**.

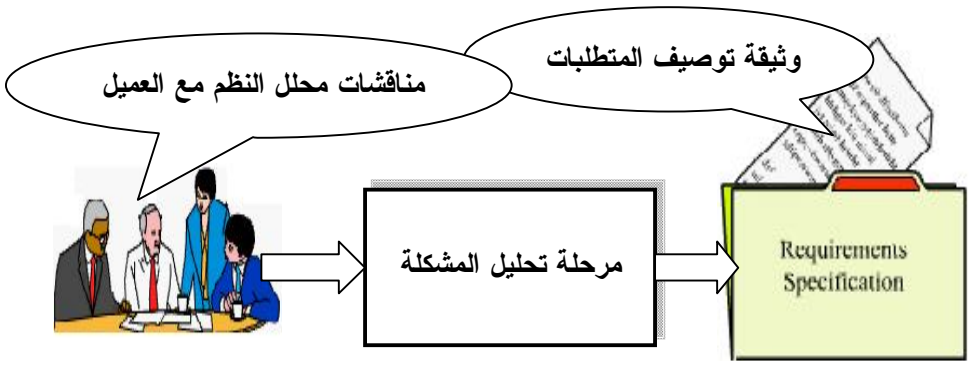
ويأتي بعد كل هذه المراحل أهم مرحلة و أطولها وأكثرها تكلفة وهي :

4. مرحلة صيانة وترقية البرنامج **(Maintenance Phase)**.

وتُعد نتائج كل مرحلة من هذه المراحل معطيات للمرحلة التي تليها ، فمثلاً نتائج مرحلة تحليل المشكلة هي معطيات لمرحلة التصميم ، ونتائج مرحلة التصميم هي معطيات لمرحلة التنفيذ ، وهكذا وسوف نتناول الآن هذه المراحل بإيجاز وبدون الدخول في التفاصيل الخاصة بكيفية إنجاز كل مرحلة ، حيث إننا سوف نتناول معظم هذه المراحل بالشرح والتفصيل في باقي أجزاء هذا الكتاب.

1. المرحلة الأولى: مرحلة تجميع المتطلبات

تركز هذه المرحلة (أو العملية) على دراسة متطلبات حل المشكلة ، حيث إنها تُعد مرحلة التوصيف الدقيق للمشروع البرمجي وتهيئته لمرحلة التصميم النهائي، ويتم ذلك عبر إعداد مجموعة من المخططات (Charts) والنماذج (Models) واللوائح (Lists) من خلال طرح مجموعة من الأسئلة المتخصصة من قبل محلل النظم للعميل لمعرفة كافة احتياجات وشروط المشروع البرمجي ، وذلك طبقاً لمنهجية محددة تسمى منهجية التحليل، وتنتهي هذالمرحلة بكتابة وثيقة رسمية **(Formal Document)** تشتمل على المواصفات الفنية للمشروع البرمجي بناءاً على متطلبات العميل مثل نطاق المعلومات الخاص بالمشروع **(Information Domain)** و الأداء الوظيفي **(Functionality)** ، الربط البيني بين البرمجية والمستخدمين **(User Interface)** ،و مستويات الأمن المطلوبة **(Security Levels)** ، بالإضافة إلى المتطلبات الاستثنائية **(Exceptional Requirements)**. ويطلق على هذه الوثيقة "وثيقة توصيف المتطلبات " **(Requirements Specification Document)** ، والتي ستكون بمثابة العقد النهائي بين مطوري المشروع البرمجي والعميل. والشكل رقم 2.1 يمثل شكل توضيحي لهذه المرحلة ، حيث إن معطيات هذه المرحلة تتمثل في متطلبات العميل ونتائجها تتمثل في كتابة "وثيقة توصيف المتطلبات".



الشكل رقم 2.1 : شكل توضيح لمرحلة تحليل المشكلة.

وبصفة عامة تتضمن مرحلة (أو عملية) تحليل المشكلة التعريف بالمشكلة من خلال العديد من النواحي والتي من أهمها :

- المدخلات **(Inputs)**: يجب على المحلل معرفة الطرق التي يفضلها العميل في عملية إدخال البيانات إلى النظام ، فهل عملية إدخال البيانات سوف تتم عن طريق القراءة من ملف ، أم سوف تكون تفاعلية بين المستخدم والنظام عن طريق قوائم اختيار بواسطة الفأرة ، أو تكون تفاعلية ولكن بطريقة مفتوحة عن طريق لوحة المفاتيح ، أو غيرها من طرق إدخال البيانات.
- المخرجات **(outputs)**: يجب – أيضاً – على المحلل الاهتمام بمعرفة متطلبات العميل من حيث شكل وهيئة نماذج المخرجات (تقارير النظام) وصيغها المختلفة ، هل تكون على شكل أرقام فقط أو أرقام وحروف ، أو خليط من الأرقام والحروف والرسومات ، أو ما شابه ذلك.
- المعالجة **(Processing)** : يجب أن يدقق المحلل في كيفية عملية المعالجة من حيث نوعية البيانات والحسابات والقرارات التي سوف تعمل على هذه البيانات لإنجاز المخرجات المطلوبة ، وكذلك من حيث بيئة التشغيل التي سوف يتم تشغيل النظام من خلالها (نظام التشغيل **(Operating System)** ، وهل عملية التشغيل سوف تتم على حاسب آلي شخصي **(Personal Computer)** ، أو من خلال استخدام الشبكات المحلية **(LANs)**.

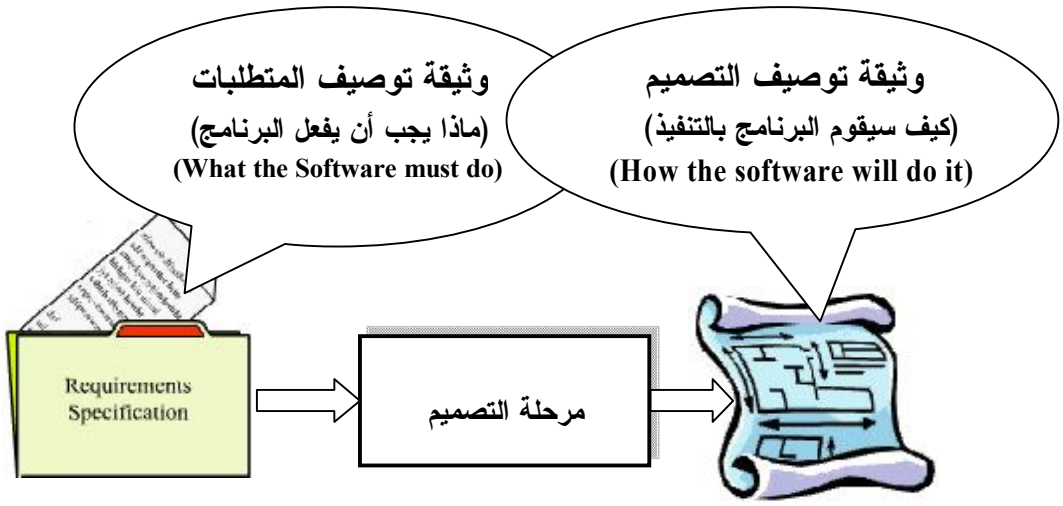
2. المرحلة الثانية مرحلة التطوير

تنقسم مرحلة التطوير إلى مرحلتين هما :

1.2 مرحلة التصميم

تُعد وثيقة توصيف متطلبات العميل هي معطيات هذه المرحلة (أو العملية) ، حيث يتم توزيع المهام على المختصين من فريق العمل من المصممين ، الذين يقومون في هذه المرحلة بتحديد الخوارزميات التي سوف يستخدمونها ، وكذلك رسم مخططات التصميم التي تتناسب مع المتطلبات المتفق عليها سابقا مع العميل ، فهناك العديد من القوالب والنماذج يتم التصميم على أساسها ، فتصنف بعضها حسب تحليل البيانات وعرضها ، والبعض حسب التسلسل الزمني أو الفترة الزمنية المحددة ، وأخرى على حسب بيئة التصميم وغيرها، وتُعد خريطة التدفق للبيانات **(Data Flowchart)** إحدى أهم الطرق المستخدمة في مرحلة التصميم ، حيث تستخدم لتوضيح المدخلات والمخرجات وتسلسل العمليات والمعدات المستخدمة لحل المشكلة، وإذا كانت المشكلة معقدة وتتعلق بأقسام عديدة تخص العميل فإنه يتم إعداد ما يسمي بخريطة التدفق للنظام **(System Flowchart)** والتي توضح العلاقات المتشعبة التي تربط بين هذه الأقسام المختلفة ونظام تدفق البيانات بين هذه الأقسام بعضها البعض ، وعلي هذه الخريطة يتم إيضاح محطات العمل والأفراد العاملين والمعدات وشكل الوثائق والأقسام المؤثرة في هذه الخريطةو يمكن للمبرمج استخدام خريطة التدفق المختصرة أو خريطة التدفق المفصلة و أيضا من الأدوات التي يمكن للمصمم استخدامها لشرح التعليمات المكونة للخوارزمية قائمة القرارات **(Decision Table)** التي يتم فيها بيان العلاقات المختلفة وأسباب ومكان التفريع للعمليات هذا وبعد إتمام عملية التصميم يتم تسجيل كل ما يتعلق بهذه المرحلة في " وثيقة توصيف التصميم" **(Design Specification Document)** ، مثل تحديد مكونات البرمجية والبناء الهيكلي للبرميجة و الربط البيني مع المستخدمين ،

هياكل وجداول البيانات ، والخوارزميات ، وذلك بحيث يضمن إنجاز الوظائف الأساسية لتحقيق متطلبات العميل . والشكل رقم 2.2 يوضح هذه المرحلة.



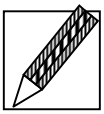
شكل 2.2 : شكل توضيحي لمرحلة التصميم.

2.2 مرحلة التنفيذ

تُعد وثيقة توصيف متطلبات التصميم هي معطيات هذه المرحلة (أو العملية) ، حيث يتم ترجمتها إلى منتج برمجي متكامل باستخدام إحدى لغات البرمجة المناسبة أو باستخدام أدوات البرمجيات المعتمدة على الحاسب **(CASE Tools)** على أيدي فريق من المبرمجين ، وتتم هذه المرحلة على خطوتين :

- الخطوة الأولي : وهي عملية البرمجة (Programming) ، حيث يتم خلالها تجزئة المشروع البرمجي إلى أجزاء تركيبية صغيرة (Modules) للتغلب على التعقيدات (Complexities) التي يمكن أن تنشأ في حالة التعامل معه ككتلة برمجية واحدة، وبعد ذلك توزع هذه الأجزاء على فريق المبرمجين ، بحيث يكون كل مبرمج مسئولاً عن برمجة واختبار وتوثيق جزئية معينة من هذه الأجزاء ، بطريقة تسمح بالتكامل مع الأجزاء الأخرى .

تدريب (1)



تتم مرحلة التنفيذ للبرمجية على خطوتين أساسيتين ، ارسم شكلاً توضيحيًا يبين كلاً من الخطوتين.

- **الخطوة الثانية : وهي عملية التكامل (Integration)** ، حيث يتم خلالها تجميع جميع أجزاء البرنامج المنفصلة **(Individual Modules)** التي تم برمجتها في الخطوة السابقة لتصبح منتجاً برمجياً متكاملاً **(Integrated Software Product).**

3. المرحلة الثالثة :مرحلة الاختبار وتشخيص الأخطاء

مرحلة (أو عملية) الاختبار ليست مرحلة منفصلة تنفذ بعد اكتمال المشروع البرمجي أو بعد انتهاء كل مرحلة فقط ، بل يجب أن تنفذ باستمرار خلال جميع مراحل تطوير المشروع البرمجي، وبصفة عامة ، تشتمل مرحلة الاختبار على خطوتين :

1.3 الخطوة الأولى :وهي خطوة التحقق (Verification)

وهي تتم في كل مرحلة من مراحل تطوير البرنامج، ويقصد بالتحقق في مرحلة معينة – هنا – تحديد إن كان البرنامج قد تم تحويله من المرحلة السابقة إلى هذه المرحلة بكفاءة وبدقة عالية وبدون أخطاء وإنه يحقق متطلبات المرحلة السابقة أم لا، فعلى سبيل المثال فإن مرحلة التنفيذ تُحول وثيقة توصيف التصميم إلى برنامج متكامل قابل للتنفيذ **(Executable Integrated Software)** ، لذا يقصد بالتحقق في هذه المرحلة تحديد إن كان البرنامج قد تم بناؤه بالشكل الذي يحقق جميع المتطلبات الموصَّفة في وثيقة توصيف التصميم وبدون أي نوع من الأخطاء أم لا، ويُعرف العاملون في مجال هندسة البرمجيات التحقق **(Verification)** بأنه إجراء جميع الاختبارات اللازمة لكل مرحلة للإجابة على السؤال التالي : هل نحن نبني البرنامج بصورة صحيحة ؟ **(Are we building the software right?)** وخطوة التحقق هنا تتم من خلال وجهة نظر المطوِّر نفسه من البرنامج.

2.3 الخطوة الثانية وهي خطوة **المصادقة (Validation)** :

وهي تتم بعد تطوير البرنامج ، ويقصد بالمصادقة – هنا – تقويم البرنامج للتأكد من أنه قد تم تصميمه بالطريقة التي يتوقعها ويرضى بها العميل وبدون أي أخطاء من أي نوع، ويُعَرف العاملون في مجال هندسة البرمجيات التحقق **(Validation)** بأنه إجراء جميع الاختبارات اللازمة للإجابة على السؤال التالي : هل نحن نبني البرنامج الصحيح ؟ **(Are we building the right software?)**، وخطوة المصادقة هنا تتم من خلال وجهة نظر العميل من البرنامج.

وعملية تشخيص الأخطاء **(Debugging)** تختلف عن عملية اختبار العيوب **(Defect Testing)** ، حيث إن عملية اختبار العيوب تهتم فقط بإثبات وجود عيوب بالبرنامج من عدمه بدون تحديد موضع أو كيفية إصلاح هذا العيب في حالة وجود عيوب. أما عملية تشخيص الأخطاء فهي تهتم بتحديد مواضع هذه العيوب وإصلاحها في نفس الوقت.

وسوف نتناول الآن أهم أنواع الأخطاء التي يمكن أن تنشأ أثناء مرحلة الاختبار وتشخيص الأخطاء ، حيث يمكن تصنيفها إلى الأنواع التالية :

❶ **أخطاء قواعد ومعاني اللغة (Syntax and Semantic Errors)** : وتنتج هذه الأخطاء عن كتابة تعليمات لا يتم فيها مراعاة قواعد ومعاني لغة البرمجة المكتوب بها البرنامج ، ويتم اكتشافها أثناء عملية ترجمة البرنامج المصدر **(Source Code)** لإنتاج البرنامج الهدف **(Object Code)** ، وذلك باستخدام مترجم اللغة **(Compiler)**.

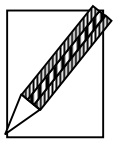
🕐 **أخطاء وقت التنفيذ (Run Time Errors)** : تظهر هذه الأخطاء أثناء وقت التنفيذ من خلال نظام التشغيل ، رغم من ترجمة البرنامج بنجاح بدون أخطاء في قواعد ومعاني اللغة، فالبرنامج قد يكون صحيحاً من ناحية قواعد ومعاني اللغة ويحتوي على جمل صحيحة ، ولكنها تسبب أخطاء عند تنفيذ البرنامج، من هذه الأخطاء مثلا القسمة علي الصفر تعطي قيمة لانهائية أو القسمة على قيمة كبيرة جداً تعطي قيمة صغيرة جداً ولا يمكن في أي لغة برمجة تمثيل هذه القيم

(Over-Flows and Under-Flows Errors) ، أو قد يكون البرنامج يحاول فتح ملف غير معَرف مسبقاً ، أو أنه قد دخل في حلقة تكرارية غير منتهية **(Open Loops) Errors**. وبمجرد اكتشاف هذه الأخطاء يقوم نظام التشغيل بوقف تنفيذ البرنامج وإعطاء رسالة تفيد بوجود الخطأ ففي هذه الحالة يجب تشغيل مشخص ومصحح الأخطاء **(Debugger)** الخاص بنفس اللغة المستخدمة لتشخيص مواضع هذه الأخطاء وتصحيحها.

- **الأخطاء المنطقية (Logical Errors) :** هي للأسف ليست أخطاء لغوية يمكن للمترجم اكتشافها أو أخطاءو في التنفيذ يمكن اكتشافها أثناء التنفيذ ، ولكنها أخطاء في تراكيب المدخلات من بيانات أو في المعادلات الرياضية ، وبالتالي تكون المخرجات غير متوقعة أو غير منطقية وغير مقبولة والسبب في هذه الأخطاء هو المبرمج الذي لا يتأكد من المدخلات أو لا يتأكد من وضع المعادلات الرياضة التي سوف تتم المعالجة علي أساسها ، أو أن يضع علامة القسمة بدلا من الضرب أو علامة الطرح بدلا من الجمع ، وخلافه.

وتُعد الأخطاء المنطقية من أصعب الأخطاء في اكتشافها ، حيث إنه في حالة وجود مثل هذه الأخطاء يجب فحص البرنامج ومطابقته مع مستندات التصميم حتى يمكن معرفة مواقع هذه الأخطاء وتصحيحها.

تدريب (2)



ما الفرق بين التحقق والمصادقة؟ أيهما يجيب عن السؤال التالي؟ :
Are we building the software right?
وأيهما يجيب عن السؤال التالي؟ :
Are we building the right software?

أسئلة تقويم ذاتي

صنف أهم أنواع الأخطاء التي يمكن أن تنشأ أثناء مرحلة الاختبار وتشخيص الأخطاء؟

4. المرحلة الرابعة: مرحلة الصيانة والارتقاء

كل المنتجات البرمجية الناجحة هي التي تُطوَر مع الوقت للتوافق مع التغيرات التي يريدها العميل ، لذا فإن عملية تطوير البرنامج لا تنتهي بتسليم المنتج النهائي للعميل ، بل تبدأ مرحلة من أهم المراحل في عمر المنتج وهي مرحلة الصيانة ومن أهم الأعمال التي قد تشملها مرحلة الصيانة ما يلي :

- معالجة الأخطاء التي قد تنشأ مع التشغيل **(Bugs Fixing)**.
- إضافة عمل وظيفي جديد للبرنامج **(Add New Functionality)**.
- إضافة تقارير خرج جديدة **(Add New Output Reports)**.
- تهيئة البرنامج للعمل على أنظمة تشغيل جديد **(Adapt the software to new Platforms)** .

5. نشاطات المظلة (Umbrella Activities)

يجرى تكامل المراحل السابقة وعملياتها المختلفة في إطار هندسة البرمجيات بعدد من نشاطات المظلة ، والتي تُجرى طوال عملية تطوير المنتج البرمجي ، ومن أهم هذه النشاطات :

1.5 إعداد الوثائق وإنتاجها

ويطلق عليها مرحلة التوثيق ، وهي ليست مرحلة منفصلة تنفذ بعد اكتمال المشروع البرمجي ، بل تبدأ هذه المرحلة مع بداية المرحلة الأولى من مراحل تطوير البرنامج إلى أن يتم تسليم كامل المنتج البرمجي للعميل (Delivery Phase) ، والتي يمكن أن تُعد أخر مرحلة في تطوير البرنامج والتي يبدأ بعدها مرحلة الصيانة.

إن التوثيق السليم للبرنامج يساعد علي الوقوف علي آخر ما تم من تعديلات وأسبابها ، وبالتالي تكون الرؤية واضحة للتعديل الجديد ؛ لذلك يجب أن يتضمن التوثيق

السليم للبرنامج ما يلي :

- ❶ تعريف المشكلة أو تحديد الهدف من البرنامج (وثيقة توصيف المتطلبات).
- وثائق ومخططات تصميم البرنامج (وثيقة توصيف التصميم).
- تسجيل الملاحظات عن الخطوات والإجراءات الهامة في شفرة البرنامج ، فهذه الملاحظات تساعد المبرمج وقارئ البرنامج على فهم البرنامج وسهولة تتبعه واكتشاف الأخطاء وتصحيحها.
- متطلبات تنفيذ البرنامج من بيئة التشغيل وحدات الإدخال والإخراج المختلفة.
- نسخة من برنامج المصدر.
- عينة من نتائج البرنامج .

2.5 متابعة المشروع البرمجي ومراقبته

تُعد إدارة المشاريع البرمجية من نشاطات المظلة ضمن هندسة البرمجيات ، حيث إنها تبدأ قبل الشروع بأية نشاط تقني وتستمر طوال تعريف البرنامج وتطويره وصيانته.

3.5 ضمان جودة المنتج البرمجي

وذلك من خلال أخذ القياسات الضرورية طوال بناء المشروع البرمجي للمساعدة في تقويم جودة المنتج البرمجي واتخاذ القرارات التصحيحية أثناء تقدم المشروع ومن أهم القياسات المباشرة لعملية هندسة البرمجيات قياس الكلفة والجهد المطبقين ،و عدد أسطر الشيفرة البرمجية المنتجةوسرعة التنفيذ و حجم الذاكرة المطلوب لعملية التنفيذ و عدد الأعطال المعلن عنها خلال فترة زمنية معينة ، هذا بالإضافة إلى قياسات المنتج غير المباشرة مثل درجة التعقيد ، والفاعلية ، والموثوقية ، وقابلية الصيانة ، وخلافه من صفات وخصائص المنتج البرمجي الجيد المذكورة في الوحدة السابقة.

4.5 إدارة تكوين البرنامج

هي نشاط مظلة يبدأ مع بداية المشروع البرمجي وينتهي فقط عند التوقف عن استخدام البرنامج ، والغرض منها هو إدارة التعديلات الحاصلة على البرنامج الذي يبنيه فريق البرمجة والتحكم فيها خلال كامل دورة حياة البرنامج بغرض تحسين السهولة التي تجرى بها التغييرات وتخفيض مقدار الجهد المبذول عندما يجب إجراء تلك التغييرات ، وذلك لزيادة الإنتاجية إلى حدها الأقصى وتقليل الأخطاء إلى حدها الأدنى، وهي أيضاً مسئولة عن تعيين هوية كل بند من بنود تكوين البرنامج (البرنامج المصدر **(Source Code)** ، والبرنامج التنفيذي **(Executable Code)** وثائق تصف البرنامج موجهة للمستخدمين والممارسين التقنيين ، وخلافه من الوثائق والبيانات الأخرى المتعلقة بالبرنامج).

5.5 إدارة المخاطر

وذلك من خلال تحديد المهام اللازمة لتقويم المخاطر التقنية والإدارية وفهمها والقيام بالإجراءات المناسبة لها في كل مرحلة من مراحل تطوير المنتج البرمجي.

أسئلة تقويم ذاتي

1. يتم تكامل مراحل تطوير البرمجيات وعمليات المختلفة بعدد من النشاطات التى تجرى طوال عملية تطوير المنتج البرمجى والتى يطلق عليها "نشاط المظلة" ما هى أهم هذه النشاطات.

2.وضع (√) أمام الإجابة الصحيحة وعلامة (X) أمام الإجانة الخطأ

- علم هندسة البرمجيات علم يهدف إلى إنتاج برمجة خاليةمن الأخطاء و ذات جوده عالية في وقت محدد بميزانيهة محددة وبطريقة اقتصادية بحيث يفي بجميع متطلبات الجهةالمطورة.
- الاتصالات بالعميل يعد نشاطاً هيكلياً يحتاج إلى مجموعة من المهام منها (إعدادجداول أعمال للاجتماع الرسمى مع العميل ، تجميع متطلبات العميل ، مراجعة متطلبات العميل،....،....)
- نشاطات المظلة وهى عباره عن مجموعة من الأنشطة مستقلة على أى نشاط برمجى هيكلى وتجرى طوال عملية بناء المنتج البرمجي.
- مرحلة تجميع المتطلبات تعد مرحلة التوصيف الدقيق للمشروع البرمجى وتهيئة لمرحلة التصميم النهائى.
- الوثيقة الرسميه تشتمل على المواصفات الفنية للمشروع البرمجى بناء على متطلبات العميل.
- وثيقة توصيف المطلبات هى معطيات مرحلة التنفيذ.

- فى مرحلةالتصميم يتم تحديد الخوارزميات التى تستخدم وكذلك رسم مخططات التصميم التى تتناسب مع المعطيات
- وثيقة توصيف التصميم تعتبر معطيات أساسية لمرحلةالتنفيذ.
- فى كل الأحوال يمكن ترجمة وثيقة النصميم إلى منتج برمجى باستخدام CASE Tools
- مرحلة الاختيار وتشخيص الأخطاء مرحلة منفصلة تنفذ بعد اكمال المشروع البرمجى.
- ليس هناك فرق على الإطلاق بين عملية تشخيص الأخطاء (Debugging) وعملية اختبار العيوب (Defect Test)
- تعتبر الأخطاء المنطقية من أسهل أنواع الأخطاء التي يمكن اكتشافها .

3. رتب المفردات التالية ترتيبا تصاعديا بحيث تمثلاً صحيحاً لمرحلة التصميم:

- **قائمة القرارات.**
- **خريطة التدفق للنظام**
- **وثيقة توصيف المتطلبات**
- **وثيقة توصيف التصميم.**

الخلاصة

عزيزي الدارس ، في نهاية هذه الوحدة دعنا نستعرض ما ورد فيها من مفاهيم:

أولاً مرحلة تحليل المشكلة وهي مرحلة توصيف دقيق للمشروع البرمجي وتهيئة لمرحلة التصميم النهائي ونتاج هذه المرحلة يتمثل في كتابة "وثيقة توصيف المتطلبات" .

ثانياً مرحلة التطوير وتشمل مرحلتي التصميم والتنفيذ حيث تعد وثيقة توصيف متطلبات العميل هي معطيات هاتين المرحلتين.

مرحلة الاختبار وتشخيص الأخطاء وهي مرحلة مستمرة خلال جميع مراحل تطوير المشروع البرمجي وتشمل خطوة التحقق وهنا يحدد ماإذاكان البرنامج قد تم بناؤه بالشكل الذي يحقق جميع المتطلبات الموضوعة في "وثيقة توصيف المتطلبات" ، الخطوة الثانية المصادقة أي تقويم البرنامج للتأكد من أنه قد تم تصميمه بالطريقة التي يتوقعها ويرضى بها العميل.

مرحلة الصيانة والارتقاء أهم المراحل في عمر المنتج وتشمل معالجة الأخطاء التي قد تنشأ مع التشغيل و إضافة عمل وظيفي جديد للبرنامج و إضافة تقارير خروج للبرنامج و تهيئة البرنامج للعمل على أنظمة تشغيل جديدةونشاطات المظلة وهي مجموعة أنشطة مستقلة عن أي نشاط برمجي هيكلي وتجري طوال عملية بناء المنتج البرمجي و من أهمهاإعداد الوثائق وإنتاجها و متابعة المشروع البرمجي ومراقبته و ضمان جودة المنتج البرمجي و إدارة تكوين البرنامج و إدارة المخاطر.

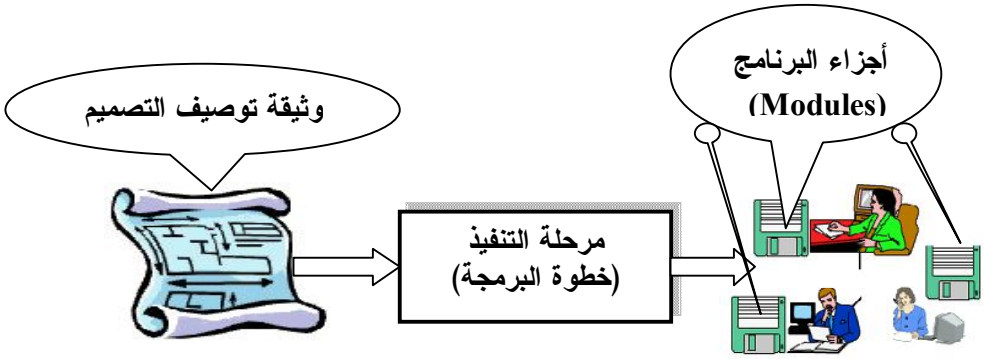
لمحة مسبقة عن الوحدة التالية

عزيزي الدارس ،،

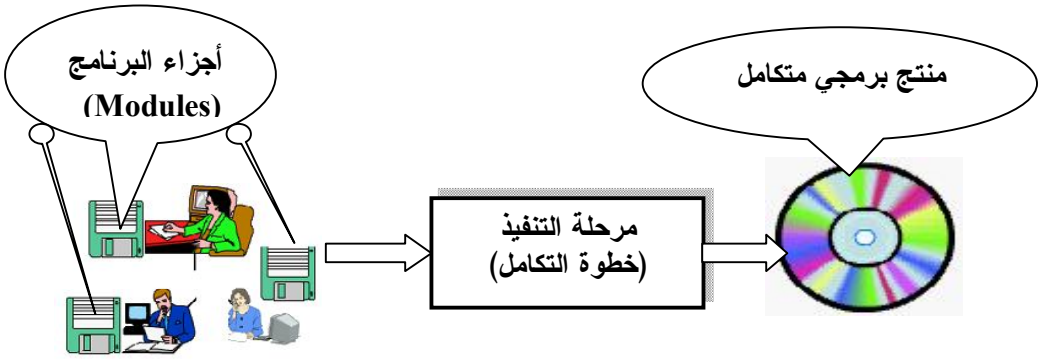
في الوحدة التالية سنتناول موضوع نماذج عمليات تطوير البرمجيات والتي سوف نتناول فيها أهم هذه النماذج بطريقة تساعدك على عمل مقارنة بينها ، ومن ثم اختيار المناسب منها بناءاً على نوعية متطلبات المشروع البرمجي المطلوب تطويره ، وكذلك القيود المفروضة عليه. نرجو أن تكون وحدة مفيدة لك.

إجابات التدريبات

تدريب (1)



شكل توضيحي لخطوة البرمجة من مرحلة التنفيذ.



شكل توضيحي لخطوة التكامل من مرحلة التنفيذ.

تدريب (2)

خطوة **التحقق (Verification)** : وهي تتم في كل مرحلة من مراحل تطوير البرنامج. ويقصد بالتحقق في مرحلة معينة – هنا – تحديد ماإذا كان البرنامج قد تم تحويله من المرحلة السابقة إلى هذه المرحلة بكفاءة وبدقة عالية وبدون أخطاء وإنه يحقق متطلبات المرحلة السابقة أم لا، وخطوة التحقق هنا تتم من خلال وجهة نظر المطوِّر نفسه من البرنامج.

خطوة **المصادقة (Validation)** : وهي تتم بعد تطوير البرنامج ، ويقصد بالمصـــادقة – هنا – تقويم البرنامج للتأكد من إنه قد تم تصميمه بالطريقة التي يتوقعها ويرضـــى بهـــا العميل وبدون أي أخطاء من أي نوع. وخطوة المصادقة هنا تتم من خلال وجهة نظـــر العميل من البرنامج.

وتجيب خطوة التحقق على السؤال التالي:

Are we building the software right?

وتجيب خطوة المصادقة على السؤال التالي:

Are we building the right software?

مسرد المصطلحات

نشاطات المظلة :

وهي عبارة عن مجموعة من الأنشطة مستقلة عن أي نشاط برمجي هيكلي وتُجرى طوال عملية بناء المنتج البرمجي (مثل : متابعة المشروع البرمجي ومراقبته ، ضمان جودة المنتج البرمجي ، إدارة تكوين البرنامج ، إدارة المخاطر ، وخلافه) ،

وثيقة توصيف المتطلبات Requirements Specification Document:

تشتمل على المواصفات الفنية للمشروع البرمجي بناءاً على متطلبات العميل.

خريطة التدفق للبيانات (Data Flowchart):

إحدى أهم الطرق المستخدمة في مرحلة التصميم ، حيث تستخدم لتوضيح المدخلات والمخرجات وتسلسل العمليات والمعدات المستخدمة لحل المشكلة.

قائمة القرارات Decision Table:

هي التي يتم فيها بيان العلاقات المختلفة وأسباب ومكان التفريع للعمليات.

التحقق Verification :

هو إجراء جميع الاختبارات اللازمة لكل مرحلة للإجابة على السؤال التالي : هل نحن نبني البرنامج بصورة صحيحة ؟ (Are we building the software right?) وخطوة التحقق هنا تتم من خلال وجهة نظر المطوِّر نفسه من البرنامج.

المصادقة Validation:

هو إجراء جميع الاختبارات اللازمة للإجابة على السؤال التالي : هل نحن نبني البرنامج الصحيح ؟ (Are we building the right software?) ، وخطوة المصادقة هنا تتم من خلال وجهة نظر العميل من البرنامج.

المصطلح بالإنجليزية	معناه بالعربية
Adapt the software to new Platforms	تهيئة البرنامج للعمل على أنظمة تشغيل جديدة
Add New Functionality	إضافة عمل وظيفي جديد للبرنامج
Bugs Fixing	معالجة الأخطاء التي قد تنشأ مع التشغيل
Charts	المخططات

المصطلح بالإنجليزية	معناه بالعربية
Common Process Framework	هيكل عام للعملية
Compiler	مترجم اللغة
Complexities	التعقيدات
Data Flowchart	خريطة التدفق للبيانات
Debugging	عملية تشخيص الأخطاء
Delivery Phase	تسليم كامل المنتج البرمجي للعميل
Design Phase	مرحلة التصميم
Design Specification Document	وثيقة توصيف التصميم
Development Phase	مرحلة التطوير
Exceptional Requirements	المتطلبات الاستثنائية
Formal Document	وثيقة رسمية
Framework Activities	نشاطات الهيكل
Functionality	الأداء الوظيفي
Implementation Phase	مرحلة التنفيذ
Information Domain	نطاق المعلومات الخاص بالمشروع
Integration	عملية التكامل
Lists	القوائم
Logical Errors	الأخطاء المنطقية
Maintenance Phase	مرحلة صيانة وترقية البرنامج
Models	النماذج
Modules	تركيبية صغيرة
Open Loops Errors	حلقة تكرارية غير منهية
Problem Analysis Phase	مرحلة تحليل المشكلة
Processing	المعالجة
Programming	عملية البرمجة
Requirements Specification Document	وثيقة توصيف المتطلبات

المصطلح بالإنجليزية	معناه بالعربية
Risk Management	إدارة المخاطر
Run Time Error	أخطاء وقت التنفيذ
Security	مستويات الأمن المطلوبة
Software Live Cycle (Processes)	دورة حياة (عمليات) البرمجيات
Syntax and Semantic Errors	أخطاء قواعد ومعاني اللغة
System Flowchart	بخريطة التدفق للنظام
Testing and Debugging Phase	مرحلة الاختبار وتشخيص الأخطاء
Umbrella Activities	نشاطات المظلة
User Interface	الربط البيني بين البرمجية والمستخدمين
Validation	المصادقة

المـراجع

أولاً : المراجع العربية

[1] أحمد شعبان دسوقي وآخرون ، "أساسيات الحاسب الآلي وتطبيقاته في التعلـــيم" ، مكتبة الرشد ، الرياض ، الطبعة الأولى ، 1427هـــ – 2006م.

ثانياً : مواقع على شبكة الإنترنت تم الاستفادة منها :

[2] http://courses.cs.vt.edu/~csonline/SE/Lessons/

[3] http://scitec.uwichill.edu.bb/cmp/online/

[4] www.bit.lk/information/tv/s2/2003/IT2401c.ppt

[5] http://www.cs.mu.oz.au/341/Lectures

[6] http://www.arabteam2000-forum.com

[7] www.cise.ufl.edu/class/cen5035/lecture_notes.html

الوحدة الثالثة

نماذج عمليات
تطوير البرمجيات

محتويات الوحدة

المقدمة

تمهيد

عزيزي الدارس، أهلاً بك إلى الوحدة الثالثة من مقرر " هندسة البرمجيات(1)" وهـذه الوحدة تتناول نماذج عمليات تطوير البرمجيات.

في هذه الوحدة سنتعرف على عدد من النماذج وهي:

نموذج الشلال أو "التتابع الخطي أو "دورة الحياة التقليدية" وسوف نتابع مع هذا النموذج المراحل التي مرت بنا في الوحدة السابقة ، ثم نوضح أهم مميـزات هـذا النمـوذج وكذلك العيوب التي يتصف بها.

نموذج النمذجة الأولية : وهنا سنتعرف على فكرة هذا النموذج ومتى يتم اللجوء إليـه ، وسنتعرف على كيفية التغلب على بعض المشاكل التي تجعل هذا النموذج منهجاً فعـالاً في هندسة البرمجيات ، وسنحدد أهم فوائد هذا النموذج.

نموذج التطوير السريع للتطبيقات: سنتعرف على الفكرة التي يعتمد عليها هذا النمـوذج وأهم العقبات التي تقف أمام استخدام هذا النموذج ، كما سنوضح في هـذا القسـم مـن الوحدة أهم نشاطات هذا النموذج.

النموذج التزايدي: يجمع هذا النموذج عناصر نموذج الشلال التتابعية بصورة تكراريـة أي أنه يقوم بتنفيذ عناصر نموذج الشلال ، وسوف نتعرف على فوائد هذا النموذج.

النموذج الحلزوني:وهو من النماذج الارتقائية ، وسنعرض هنا بعض الأمثلـة لكـوارث خلال مشاريع تطوير البرمجيات ، وسنتعرف على أهم مميزات هذا النمـوذج ، كمـا سنتعرف أيضاً على الوثيقة التي تصف بشكل عام غير مفصل كيـف يجـب أن يعمـل النظام "وثيقة مفهوم العمليات" .

نموذج الطرق المنهجية: ويشمل هذا النموذج مجموعة الأنشـطة التـي تمكِـن فريـق التطوير من وصف متطلبات النظام والتحقق من صحة عملة عن طريق تدوين رياضي دقيق، وسوف نتناول أهم مميزات هذا النموذج وسلبياته.

تجد عزيزي الدارس فى ثنايا الوحدة بعض التدريبات التي يساعدك تنفيذها على فهم محتوى المادة هذا بالإضافة إلى أسئلة التقويم الذاتي التي تهدف إلى ترسيخ الفهم وتعزيزه لديك و تعميقه .

أهداف الوحدة

عزيزي الدارس،

بنهاية دراسة هذه الوحدة ينبغي أن تكون قادراً على أن :

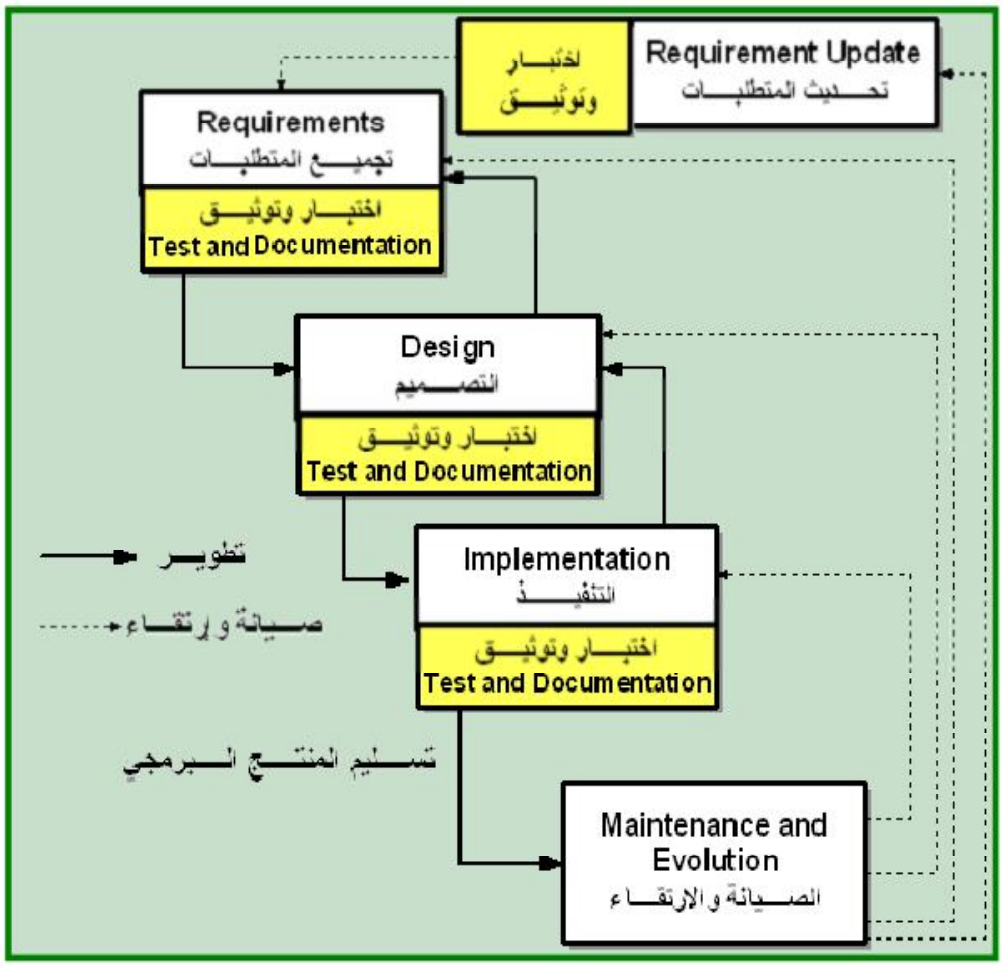
- تعبر عن أهمية استخدام نماذج تطوير عمليات البرمجيات لتطـــوير البرمجيات.
- تحدد الفروق المختلفة بين نماذج تطوير عمليات البرمجيات.
- تختار النموذج المناسب لتطوير منتج برمجي طبقاً لمعطيـــات هـــذا المنتج.
- تطبق المبادئ الهندسية الصحيحة على عملية تطوير البرمجيات.
- تصف الأنشطة الخاصة بكل نموذج من النماذج التـــي تـــم تناولهـــا وكيفية تنفيذها.
- تعدد أهم مميزات وعيوبه كل نموذج من النماذج التي تـــم تناولهـــا وعيوبه.
- تحدد متى وكيف يستخدم كل نموذج من النماذج التي تم تناولها .
- تشرح أهم النماذج المستخدمة في عملية تطوير البرمجيات.
- تسمي أهم التحديات الرئيسية التي تواجه عملية تطوير البرمجيات.
- تعدد المخاطر الرئيسية التي تواجه عملية تطوير البرمجيات.

توطئة

نموذج عمليات تطوير البرمجيات هو مجرد وصف أو تصور بسيط (إستراتيجية تطوير) لتمثيل أنشطة تطوير عمليات البرمجيات والتي تقدم من وجهة نظر محددة ، بحيث تؤدي في النهاية إلى إنتاج منتج برمجي كامل قابل للتنفيذ، وسوف نتناول في الأجزاء التالية أهم هذه النماذج بطريقة تساعدك عزيزي القارئ على عمل مقارنة بينها ، ومن ثم اختيار المناسب منها بناءاً على نوعية متطلبات المشروع البرمجي المطلوب تطويره ، وكذلك القيود المفروضة عليه.

1. نموذج الشلال (Waterfall Model)

يقترح **نموذج الشلال** ــ والذي يسمى أحياناً " بالنموذج التتابعي الخطي " ، وأحياناً أخرى " بنموذج دورة الحياة التقليدية " ـــ منهجاً تتابعياً منظماً لتطوير البرمجيات ، حيث يبدأ من مرحلة " تحليل المتطلبات " وينتهي بمرحلة " الصيانة والارتقاء " مروراً بباقي المراحل التقليدية لدورة حياة البرمجيات التي ذكرناه سابقاً. وكما هو موضح بالشكل رقم 3.1 ، يوفر نموذج الشلال الأصلي ، الذي أقترحه **Winston Royce** ، حلقات تغذية رجعية ، إلا أن غالبية المؤسسات البرمجية التي تطبق هذا النموذج تعامله وكأنه تتابعي خطي تفادياً للمشاكل والعقبات التي سوف تواجههم في حالة أخذ نواتج حلقات التغذية الرجعية في الحسبان أثناء عمليات التطويرو يتكون نموذج الشلال من النشاطات العامة التي تم تتناولها سابقاً ، وهي ، كما هو موضح بالشكل رقم 3.1 ، كما يلي :



شكل 3.1 : رسم توضيحى لنموذج الشلال لتطوير البرمجيات

مرحلة تجميع المتطلبات : وكما تم توضيحه سابقاً ، فالغرض منها هو : توثيق متطلبات النظام والبرنامج ومراجعتها مع العميل قبل بداية عملية التصميم.

مرحلة التصميم : في هذه المرحلة ، وكما تم توضيحه سابقاً ، يتم تحويل وثيقة توصيف متطلبات العميل إلى وثيقة توصيف التصميم ، والتي تتضمن ــ في الغالب ــ بنية البيانات (Data Structure) ، وبنيان البرمجية (Software Architecture) ،

وتفاصيل الخوارزميات (Algorithmic Detail) التي سوف تستخدم في عملية التشفير ، وواجهة الربط البنية مع المستخدم (User Interface). ويتم تقويم وثيقة توصيف التصميم والتأكد من أنها تتطابق مع معايير الجودة المطلوبة وخالية من الأخطاء قبل البدء في مرحلة التنفيذ، وفي هذه المرحلة ــ أيضاً ــ يتم تحديد متطلبات الأجهزة التي تتناسب مع توصيف البرمجية التي سوف يتم تنفيذها.

مرحلة التنفيذ : في هذه المرحلة ، وكما تم توضيحه سابقاً ، يتم تحويل وثيقة توصيف التصميم إلى منتج برمجي متكامل قابل للتنفيذ.

مرحلة الاختبار وتشخيص الأخطاء و في هذه المرحلة ، وكما تم توضيحه سابقاً ، يتم اختبار البرمجية حال توليد الشيفرة البرمجية والتأكد من خلو البرمجية من أخطاء منطقية العلاقات الداخلية ، وكذلك الخارجية ، أي ضمان أن الدخل المعرف سيعطي نتائج فعلية تتوافق مع النتائج المطلوبة.

مرحلة الاختبار والارتقاء و في هذه المرحلة ، وكما تم توضيحه سابقاً ، يتم صيانة وتغيير بعض من أجزاء النظام البرمجي تجاوباً مع المتغيرات في متطلبات العميل.

يتميز نموذج الشلال بالبساطة في الفهم والإدارة حيث إنه من السهل على المطوِّر توضيح كيفية سير العمل في المشروع البرمجي للعميل ، وكذلك يتميز بأن عمليات الاختبار **(Test)** وعملية التوثيق **(Documentation)** تتم بصفة ملازمة مع تنفيذ كل مرحلة من مراحل تطوير البرمجية، ولكن وعلى الرغم من أن نموذج الشلال هو الأكثر استخداماً في هندسة البرمجيات إلا أن لهذا النموذج العديد من العيوب منها :

- يتطلب هذا النموذج تحديد متطلبات العميل بكل دقة وأن تكون طريقة الحل واضحة ومفهومة وسبق تحليل النظام بالكامل قبل البدء في عملية التصميم ، حيث إن هذا

النموذج يفتقر إلى عمليات التكرار والإعادة، ولهذا تبرز صعوبات عند استخدام هذا النموذج مع الأنظمة الضخمة لعدم التحديد الدقيق للمتطلبات والتي تُعد ظاهرة عامة في مثل هذه الأنظمة.

- تأخر صدور النسخة العاملة من البرمجية حتى وقت متأخر من الجدول الزمني للمشروع.
- صعوبة إجراء التعديلات في المراحل الأولية في حالة تقدم المشروع إلى المراحل النهائية ، ويكون الخطأ شبه كارثة إذ لم يُكتشف إلا عند مراجعة المنتج النهائي من البرمجية.
- اعتماد المراحل (النشاطات) المختلفة على بعضها البعض يؤدي إلى حالات انتظار بين بعض من أعضاء فريق المشروع ، بمعني انتظار بعض من أعضاء فريق المشروع وجلوسهم بدون عمل حتى يُنهي البعض الآخر مهام مترابطة وضرورة لهم لبدء عملهم، مما يؤدي ذلك إلى تأخر تسليم المشروع.

أسئلة تقويم ذاتي

اذكر أهم مميزات وعيوب نموذج الشلال لتطوير البرمجيات.

2. نموذج النمذجة الأولية (Prototyping Model) :

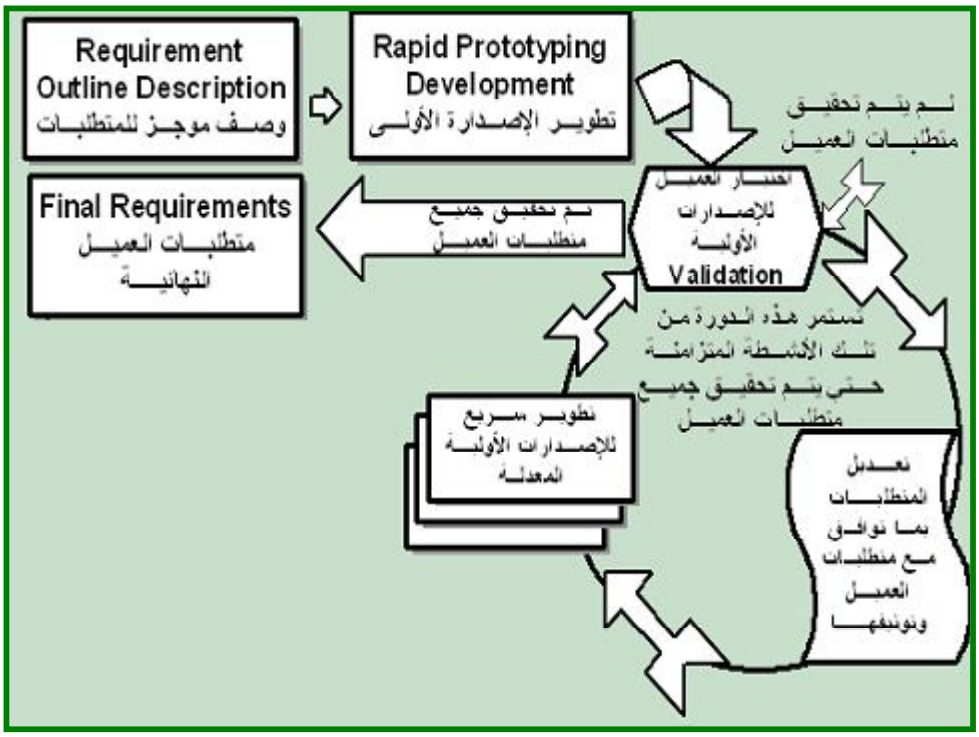
يعتمد فكرة هذا النموذج على التطوير السريع **(Rapid Development)** لنماذج تحاكي شكلياً البرنامج المطلوب أثناء فترة تحديد متطلبات العميل للتأكد من جدوى البرنامج وفهم متطلباته. إن الفرق الوحيد بين النماذج الأصلية والنظام المرغوب فيه هو كفاءة النماذج وكمية المعلومات التي تتعامل بها.

ويتم اللجوء إلى هذا النموذج عندما لا تتوافر متطلبات كافية ومحددة عند العميل لبناء البرنامج ولكنه يعرف مجموعة من الأهداف والخطوط العامة للبرنامج **(Outline Specifications)**، أي أن هذا النموذج يستخدم ــ مثالياً ــ كآلية لتحديد متطلبات

البرنامج قبل البدء في تطوير الإصدارة النهائية من البرنامج و في هذه الحالة يطلق على هذا الأسلوب : "أسلوب النمذجة الأولية الملقاة جانباً " **(Throw-Away Prototyping Technique)** ، حيث إنه يتم إلقاء أخر نموذج من النماذج الأولية جانباً بعد تحقيق الهدف المنشود وهو الحصول على المتطلبات النهائية للعميل في صورة مواصفات واضحة المعالم.

كما هو واضح من الشكل 3.2 التالي ، يبدأ التطوير السريع للإصدارة الأولى الأولية **(First Prototype Version)** بناءاً على وصف موجز لمتطلبات العميل المعروفة لديه ، حيث يركز التطوير السريع على تمثيل نواحي محددة من البرنامج وخاصة تلك التي تكون مرئية للعميل ، ومن ثم يتم إعادة الإصدارة الأولى الأولية إلى العميل لتقويمها ولإضافة متطلبات إضافية ، ومن ثم تتم دورة تطوير الإصدارات الأولية المعدلة **(Intermediate Prototype Versions)** لتتواءم مع متطلبات العميل الإضافية وتقويمها من العميل وتحديث المتطلبات مرة أخرى بإضافة متطلبات العميل الجديدة إليها

ويستمر تكرار هذه العملية حتى يتم تحقيق جميع متطلبات العميل ويرضى عن الأداء الوظيفي لإحدى الإصدارات الأولية، ومن ثم يتم بناء البرنامج لتحقيق المتطلبات النهائية للعميل بناء على الدورة التقليدية للبرمجيات والتي تم شرحها سابقاً، وتُعد سرعة وتكلفة بناء النماذج الأولية، أمرا هاما في هذا الأسلوب، تحقيق السرعة العالية والتكلفة القليلة لبناء هذه النماذج يتم بتجاهل متطلبات الجودة للمنتج النهائي.



شكل 3.2 : شكل توضيحي لنموذج النمذجة الأولية لتطوير البرمجيات

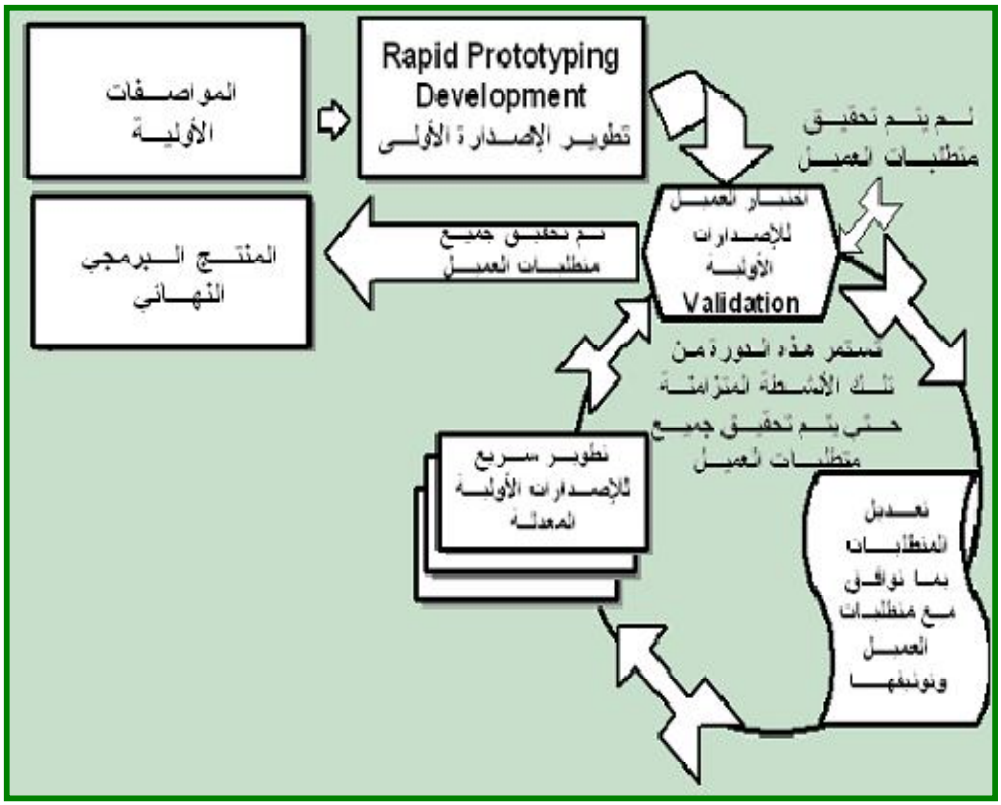
ومن عيوب **أسلوب النمذجة الأولية الملقاة جانباً** أنه بعد الانتهاء من تحديد متطلبات العميل النهائية لا يتم الاستفادة من الإصدارات الأولية التي تم تطويرها ، بل إنها ترمى جانباً ، وذلك في رأي معظم المتخصصين في هندسة البرمجيات يرجع إلى أن الإصدارات الأولية يتم تطويرها بسرعة بدون الأخذ في الاعتبار مقاييس الجودة ، حيث إن معظم المطوِّرين يقوم ببعض التجاوزات أثناء عملية التنفيذ لجعل الإصدارة تعمل بسرعة ، مثل استخدام نظام تشغيل أو لغة برمجة غير مناسبين فقط لأنهما متوفران لديهم ويُحسنون استخدامهما ، وقد يعتمدون خوارزميات غير كافية مما يتسبب في أن يكون البرنامج بطيئاً جداً ، أو صعب الاستخدام ، أو يحتاج إلى ذاكرة كبيرة لتشغيله ، وخلافه، وكذلك ورغم أن استخدام نموذج النمذجة الأولية في تطوير البرمجيات يعطي

للعميل فكرة عامة وجيدة عن النظام الفعلي ، لكن يمكن أن تحدث بعض التجاوزات من العميل أو من إدارة تطوير البرمجيات (المطوِّرين) ، فمثلاً يمكن أن يصر العميل على إجراء بعض التعديلات البسيطة على الإصدارات الأولية لكي يستخدمها كمنتج نهائي بالرغم من أنها لا تتوافق مع مقاييس الجودة المطلوبة ، وكذلك لا تتوافق مع معايير الصيانة على المدى البعيد ، وفي كثير من الأحيان تستجيب إدارة تطوير البرمجيات على ذلك، أما من ناحية المطوِّرين فقد يقنعون أنفسهم بصلاحية الخيارات التي استخدموها في تنفيذ الإصدارات الأولية وينسى أنها غير مناسبة وبالتالي يكون المنتج البرمجي ضعيف في النواحي التي استخدمت فيها تلك الخيارات.

لذا ، وللتغلب على هذه المشاكل ولجعل نموذج النمذجة الأولية منهجاً فعالاً في هندسة البرمجيات يجب أن يتفق العميل مع مؤسسة تطوير البرمجيات على أن الإصدارات الأولية من البرنامج تستخدم فقط كآلية لتحديد المتطلبات الفعلية ، ثم ترمى جانباً وتبنى البرمجيات الفعلية مع مراعاة مقاييس الجودة ومعايير الصيانة على المدى البعيد.

ويوجد أسلوب أخر من استخدام النمذجة الأولية في تطوير البرمجيات يطلق عليه نموذج " **أسلوب النمذجة الأولية الارتقائية**" **(Evolutionary Prototyping Technique)** ، وهو يشبه في شكله الخارجي "أسلوب النمذجة الأولية الملقاة جانباً" ، ولكنه يختلف عنه في آلية التنفيذ أثناء عملية تكرار تطوير النماذج الأولية ، حيث إن عملية التطوير تتم من خلال إطار عمل نظامي **(Systematic Framework)** تبدأ بعمل تحليل مبدئي لمتطلبات العميل للحصول على مواصفات أولية **(Initial Specification)** يتم على أساسها بناء أول النماذج الأولية بطريقة نظامية يراعي فيها مقاييس الجودة والأداء ، ومن ثم تتم دورة تطوير الإصدارات الأولية المعدلة **(Intermediate Prototype Versions)** لتتواءم مع متطلبات العميل الإضافية وتقيمها من العميل وتحديث المتطلبات مرة أخرى بإضافة متطلبات العميل الجديدة إليها ، ويستمر تكرار هذه العملية حتى يتم تحقيق جميع متطلبات العميل ويرضى عن الأداء الوظيفي لإحدى الإصدارات الأولية ،

والتي تُعد في هذه الحالة المنتج النهائي ، كما هو موضح بالشكل رقم 3.3.



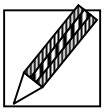
شكل 3.3 : شكل توضيحي لأسلوب النمذجة الأولية الارتقائية في تطوير البرمجيات.

ويجب بناء النموذج الاولي بطريقة سريعة حتى يتمكن المستخدمون من تقويمه وإبداء ملاحظاتهم عليه في مراحله الأولية،كما يجب تعديله أيضاً بشكل سريع لكي يتضمن آراء المستخدمين التي تقترح تعديلات تفي بمتطلباتهم، لكن كيف يمكن إنتاج البرنامج بطريقة أسرع من الطرق المعتادة ؟ ، يمكن أن يتم ذلك باستخدام عدد من الوسائل والأساليب الملائمة مثل : تكنولوجيا الجيل الرابع **(Fourth Generation Technology "4GT")** ، والأجزاء القابلة للاستخدام مرة أخرى ، حيث إنه يمكن تخفيض الوقت المطلوب لتطوير البرنامج إذا كان من الممكن إعادة استخدام أجزاء عديدة من ذلك النظام بدلاً من الأجزاء المصممة والمنفذة، كما يمكن

بناء النماذج الأولية بسرعة في حالة وجود مكتبة للمكونات التي يمكن إعادة استعمالها ، وبعض الآليات الأخرى التي تساعد على دمج هذه الأجزاء في الأنظمة. وقد تستخدم الأجزاء التي من الممكن إعادة استخدامها أيضاً في تكوين النظام النهائي ، وبهذه الكيفية يمكن تقليل تكلفة تطوير البرنامج.

- **ولكن ما هي فوائد نموذج النمذجة الأولية بصفة عامة؟** : يمكن القول إن من أهم فوائد هذا النموذج أنه أثناء تحديد متطلبات العميل يستطيع المطور أن يُطلع العميل على نظام العمل المقترح في مرحلة مبكرة جداً ، وعادةً ما يكون العملاء غير متأكدين دائماً مما يريدون من النظام أن يفعل ، ومن الصعب بالنسبة لهم توضيح المتطلبات الوظيفية بالتفصيل وبصورة واضحة قبل أن تتاح لهم الفرصة لتجربة الخيارات بشكل فعال، بالإضافة إلى أن النموذج يقدم للعميل صورة واضحة للشكل المتخيل للنظام والعمليات التي سيقوم بتنفيذها، وبواسطة فحص الخيارات بالنسبة للنسخ المختلفة من النموذج ، يشجع العملاء على اكتشاف المتطلبات التي ربما لا يكونوا قد فكروا فيها إلا بعد التنفيذ الكامل للنظام، وتكمن القيمة في الاتصال المثالي بين العميل والمحلل في استطاعة العملاء أن يخبروا المطور بآرائهم في النظام وتنفيذ التعديلات المطلوبة والتأكد من صلاحية النظام في وقت مبكر من العمر الافتراضي للمشروع، وهكذا يزيل النموذج عدداً كبيراً من أخطاء التصميم في مراحل مبكرة جداً من المشروع، ويعزي توفير الوقت والجهد إلى تجنب العمل في تغيير النظام الذي لا ينفذ ما يطلبه المستخدم فعلياً، بالإضافة إلى إنه عندما يكون العملاء مشتركين في عملية التطوير ،غالباً ما يكسبون ثقة النظام، وقد يشاهدون أولاً المشاكل والأخطاء ، لكن يمكنهم أيضاً مشاهدة الأخطاء وهي تعالج بسرعة.

تدريب(1)



لخص أهم فوائد نموذج النمذجة الأولية بصفة عامة

أسئلة تقويم ذاتي

اذكر أهم عيوب أسلوب النمذجة الملقاة جانبا.
ما هي أهم الفروق الأساسية بين أسلوب النمذجة الأولية الملقاة جانبا وأسلوب النمذجة الأولية الارتقائية؟.
لخص أهم فوائد نموذج النمذجة الأولية بصفة عامة.

3. نموذج التطوير السريع للتطبيقات

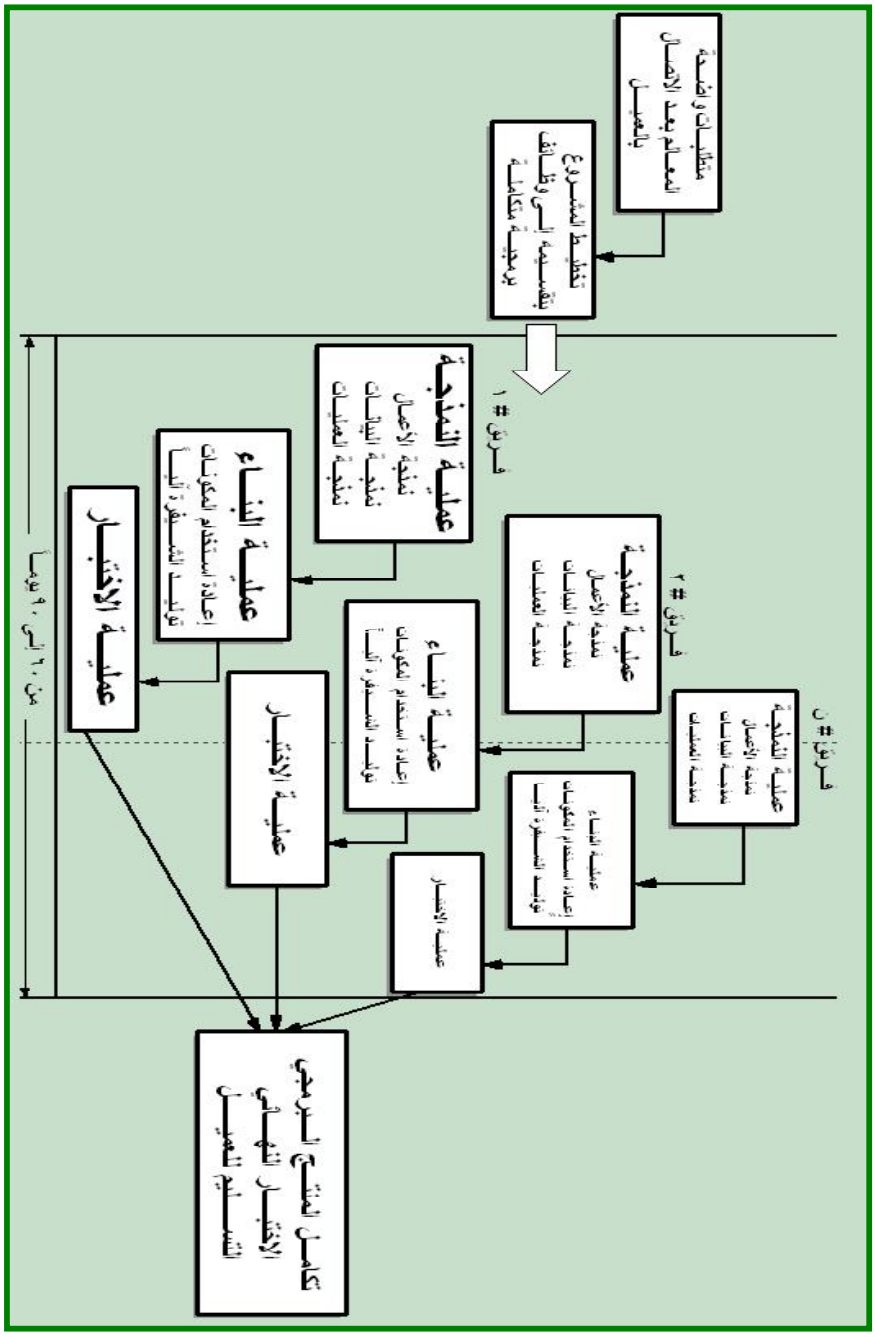
(Rapid Application Development "RAD" Model) :

يعتمد نموذج التطوير السريع للتطبيقات على تقسيم المشروع البرمجي إلى عدة وظائف برمجية رئيسية يتم تناول كل منها بواسطة فريق عمل مستقل بحيث تسمح عملية التقسيم لكل فريق بتطوير الوظيفة الخاصة به كنظام يعمل تماماً **(Fully Functional System)** خلال فترة قصيرة تتراوح من (60) إلى (90) يوماً ، ومن ثم تتكامل هذه الوظائف لتشكل كياناً واحداً ، كما موضح بالشكل رقم 3.4.

وللقيود المفروضة على زمن تطوير المشروع يُعد مبدأ إعادة استخدام المكونات البرمجية القابلة لإعادة الاستخدام ، وخصوصاً تلك التي تم اختبارها من قبل ، وكذلك تقنيات هندسة البرمجيات بمساعدة الحاسب **(CASE)** في عملية تنفيذ البرمجيات هو حجر الزاوية في نموذج التطوير السريع للتطبيقات ، هذا بالإضافة إلى عملية تقسيم المشروع إلى وظائف برمجية رئيسية كما تم ذكره.

ولاستخدام نموذج التطوير السريع للتطبيقات في تطوير الأنظمة البرمجية يجب أن تكون متطلبات العميل مفهومة جيداً ، بالإضافة إلى إمكانية تقسيم المشروع بشكل مناسب إلى وظائف برمجية رئيسية مستقلة، وفي حالة المشاريع كبيرة الحجم يتطلب نموذج التطوير السريع للتطبيقات موارد بشرية كافية لتكوين العدد المطلوب من فرق

العمل، ويُعد ما تم ذكره من أهم العقبات التي تقف أمام استخدام نموذج التطوير السريع للتطبيقات في بناء النظم البرمجية.



شكل 3.4 شكل توضيحي لنموذج التطوير السريع للتطبيقات

وتُعد تطبيقات نظم المعلومات من أهم التطبيقات التي يمكن استخدام نموذج التطوير السريع للتطبيقات في تطويرها لكونها تتميز باستقلالية بعض من وظائفها ، بالإضافة إلى إمكانية فهم وتحديد متطلبات العميل على الوجه الأمثل. والشكل رقم 3.4 يوضح أهم نشاطات نموذج التطوير السريع للتطبيقات، لتطوير تطبيقات نظم المعلومات ، وهي كمايلي :

- عملية النمذجة **(Modeling)** : وتشمل نمذجة الأعمال **(Business Modeling)** ، ونمذجة البيانات **(Data Modeling)** ، ونمذجة العمليات **(Processes Modeling)** ، وذلك لتصميم النظام من حيث تدفق الأعمال والبيانات **(Work and Data flow)** والعلاقة المترابطة بينهما لتحقيق وظائف الأعمال المطلوبة من النظام.

- عملية بناء البرمجية : وتشمل : عملية توليد التطبيق **(Application Generation)** ، باستخدام مبدأ إعادة استخدام المكونات **(Component Reuse)** ، وكذلك استخدام الأدوات المؤتمته مثل تقنيات **CASE** لتوليد الشيفرة البرمجية آلياً **(Automatic Code Generation)** مثل مطوِّر ومصمم أوراكل **(Oracle Developer and Designer)**، وذلك بدون الحاجة إلى استخدام لغات البرمجة التقليدية.

- عملية الاختبار : وتشمل : اختبار كامل البرمجية وخصوصاً المكونات الجديدة منها ، وكذلك تجربة كل الواجهات (الشاشات) تجريباً كاملاً.

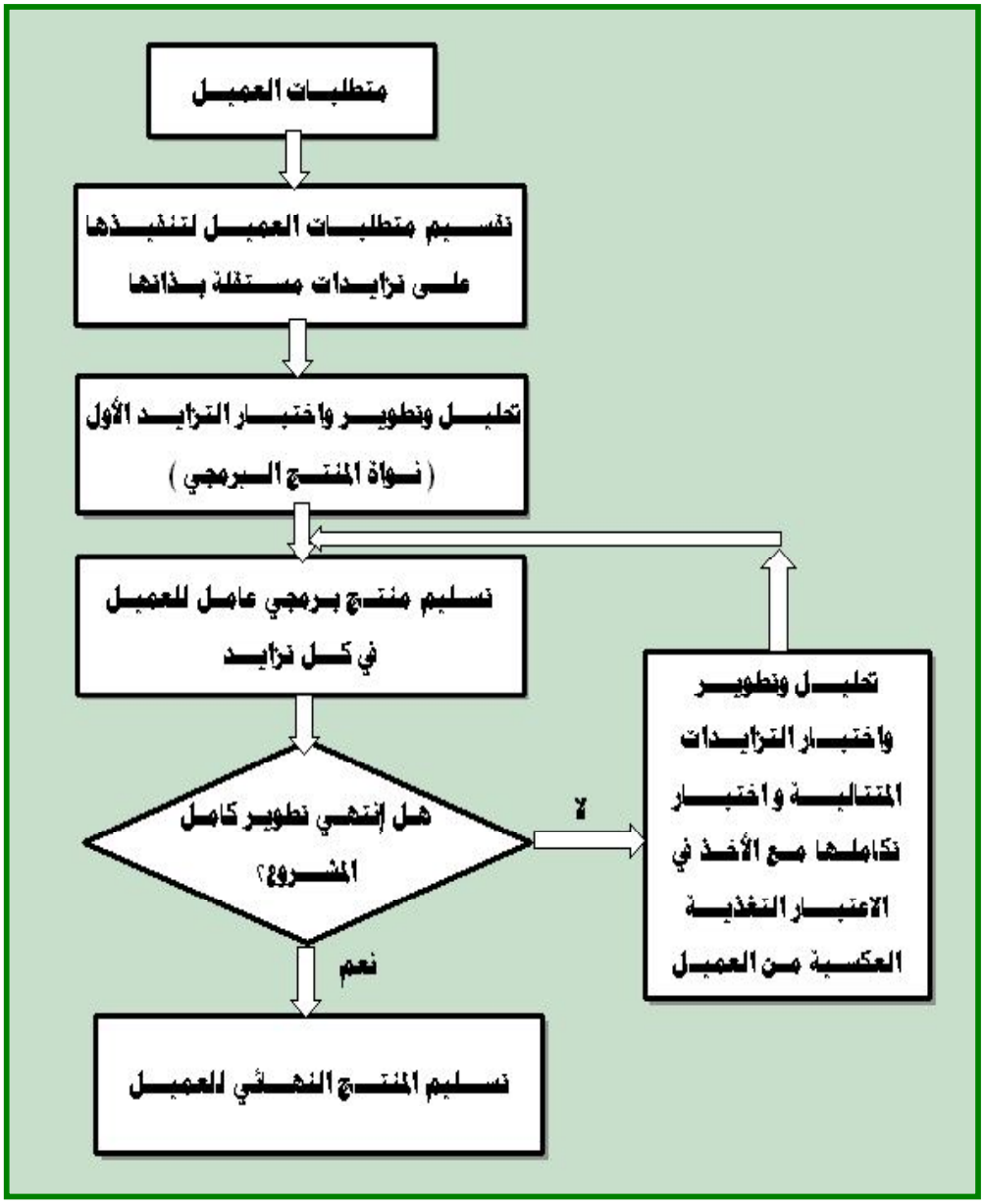
أسئلة تقويم ذاتي

ما هي أهم نشاطات نموذج التطوير السريع للتطبيقات؟.

4. النموذج التزايدي (Incremental Model) :

يُعـــد هـــذا النموذج التزايدي مـــن النمـــاذج الارتقائيـــة (التطوُّريـــة) **(Evolutionary Models)** التي لها صفة التكرارية ، حيث إنه يجمع عناصر نمـــوذج الشلال التتابعية (التحليل ــ التطوير ــ الاختبار) بصورة تكرارية ، أي إنه يقوم بتنفيـــذ عناصر نموذج الشلال كتتابعات خطية بأسلوب متعاقب بحيث ينتج في كل تتابع تزايـــداً عاملاً من البرنامج يسلم للعميل ، كما هو بالشكل رقم 3.5. وفي بداية عملية التطـــوير باستخدام النموذج التزايدي يتم تقسيم متطلبات العميل لتنفيذها على تزايدات بحيث يكون التزايد الأول يحقق المتطلبات الأساسية للعميل ، ويطلق على هذا التزايد : " نواة المنتج " **(Product Core)** . وبالرغم من أن النموذج التزايدي نموذجاً ارتقائيا مثله فـــي ذلـــك مثل نموذج النمذجة الأولية إلا إنه يسلم للعميل منتجاً برمجياً عاملاً فـــي كـــل تزايـــد بالإضافة إلى أنه يفيد المستخدم في عملية التقويم المستمر للبرنـــامج، ويكمـــن التحـــدي الأكبر لتطبيق النموذج التزايدي في تطوير البرمجيات في تجزئة المشروع إلى مكونات مستقلة بذاتها ومعرفة المكونات التي قد تعتمد على مكونات أخرى حتى يمكـــن ترتيـــب عملية تطوير التزايدات بطريقة متعاقبة مع تقدم زمن التطوير.

إن استخدام النموذج التزايدي في تطوير البرمجيات مفيـــد فـــي حالـــة تطـــوير المشاريع العملاقة وخصوصاً عند عدم توافر الكفاءات المطلوبة في شركة واحدة للقيـــام الكامل بالمشروع ، ففي هذه الحالة يمكن تقسيم المشروع ليقوم بتطـــويره العديـــد مـــن الشركات ، وأيضاً عند عدم توافر فريق كامل في الشركة الواحـــدة لإنجـــاز الأعمـــال المنوطة بها في الموعد المحدد للمشروع.



شكل 3.5 : شكل توضيحي للنموذج التزايدي لتطوير البرمجيات.

أسئلة تقويم ذاتي

أعد ترتيب المفردات التالية لتبين شكلاً توضيحياً للنموذج التزايدي لتطوير البرمجيات.

* هل انتهى تطوير كامل للمشروع؟.
* تسليم المنتج النهائي للعميل.
* تقسيم متطلبات العميل لتنفيذها على تزايدات مستقلة بذاتها.
* تحليل وتطوير واختبار التزايد الأول (نواة المنتج البر مجي).
* متطلبات العميل.
* تحليل وتطوير واختبار التزايدات المتتالية واختبار تكاملها مع الأخذ في الاعتبار التغذية العكسية من العميل.

5. النموذج الحلزوني (Spiral Model)

يُعد النموذج الحلزوني الذي اقترحه **"Barry Boehm"** في عام 1988م من النماذج الارتقائية **(Evolutionary Model)** ، وقد بدء التفكير فيه لتطوير البرمجيات من منطلق أنه من المجازفة الانتظار لفترة متأخرة من المشروع لاكتشاف الأخطاء والعمل على تصحيحها كما هو الحال في نموذج الشلال ؛ لذا فإن النموذج الحلزوني ــ في كثير من الأحيان ــ يدمج نموذج النمذجة الأولية **(Prototyping Model)** بطبيعته التكرارية في الإطار النظامي **(Systematic Framework)** لنموذج الشلال ، مع الاعتراف بوجود أشياء مجهولة أو مشكوك في صحتها **(Uncertainty)** في مستويات عديدة أثناء عملية تطوير البرمجيات (مخاطر **"Risk"**) ، وذلك لفهم هذه المخاطر والعمل على تلافيها في كل مستوى من مستويات تطوير المنتج البرمجي من خلال التحليل الدوري للمخاطر أثناء عملية التطوير. ويعرف المصطلح **"Risk"** على أنه كارثة أو مصيبة يمكن أن تحدث خلال

تطوير المشروعات البرمجية، وبالطبع فإن للكوارث درجات وتختلف من منطقة لأخرى ومن أمثلة هذه الكوارث خلال مشاريع تطوير البرمجيات ما يلي :

- إهمال بعض من متطلبات العميل أثناء عملية التطوير.
- قيام الشركات المنافسة بطرح منتج برمجي منافس في السوق.
- ترك شخص ما فريق التطوير.
- تأخر إحدى مهام مقومات التطوير عن الموعد النهائي المحدد.
- اكتشاف أن البرنامج يؤدي وظيفته ببطء شديد.
- اكتشاف أن البرنامج يشغل مساحة كبيرة من الذاكرة الرئيسية عند التشغيل.
- توفر آلية تطوير برمجيات جديدة وجيدة أثناء عملية التطوير.
- فهم بعض من متطلبات العميل فهماً خاطئاً.
- تعديل العميل بعض المتطلبات.
- تغيير شكل الجزء المادي المستهدف.
- عدم قدرة فريق التطوير من إنتاج البرمجية في الوقت المحدد ووفقا للميزانية الموضوعة.
- القدرة على إنتاج البرمجية ولكن بميزانية أعلى من المحددة و في الوقت المحدد.
- بعد جهد جهيد ووقت كبير واستغلال للموارد تم اكتشاف أنه لا يمكن تطوير البرمجية لتغطي المتطلبات عند أي تكاليف.
- ظهور تقنية جديدة **(Technological Breakthroughs)** تجعل المشروع من الطراز القديم ، أي لا فائدة منه بعد ظهور هذه التقنية الجديدة.

كما أن النموذج الحلزوني يحاول معالجة القصور الموجودة في نموذج الشلال والتي من أهمهما من ــ وجه نظر ــ مطور النموذج الحلزوني **(Boehm)** : أن نموذج الشلال لا يعمل بكفاءة في حالة استخدامه في تطوير العديد من التطبيقات وخصوصاً تلك التي تحتاج تفاعلاً مع المستخدم **(Interactive End-User Applications)** ، حيث إنه من الصعب على المستخدم (العميل) أن يحدد متطلبات تصميم هذه الأنواع من التطبيقات

وخصوصاً متطلبات تصميم وجهة الاستخدام **(Interface Design)** ، وبالتالي فإنه من المتوقع حدوث بعض من المشاكل حيث إن العقد بين العميل والشركة المنفذة للمشروع يتم قبل تحديد دقيق للمتطلبات، بالإضافة إلى ذلك فإن نموذج الشلال لا يدعم عملية إدارة وتحليل المخاطر أثناء المراحل المختلفة من عملية تطوير البرمجيات، وهذا في حد ذاته يُعد مجازفة وذلك لأن عملية اكتشاف الأخطاء والعمل على تصحيحها تتم في مراحل متأخرة من عمر المشروع .

لذا ، فإنه من أهم مميزات النموذج الحلزوني أنه يضع شروطاً واضحة للتعامل مع المهام التي تساورها الشكوك لتقليل المخاطر التي تواجه المشروع البرمجي ، حيث يتم التعامل مع هذه المخاطر بشكل متكرر في كل مرحلة من مراحل المشروع ، علاوة على ذلك يقدم النموذج الحلزوني إمكانية تطوير سريع لنسخ تزايدية من البرنامج ، حيث إنه حسب النموذج الحلزوني يتم تطوير البرنامج من خلال إصدار سلسلة من النسخ الأولية باستخدام النمذجة الأولية **(Prototyping)**. ولكن النموذج الحلزوني يُعد من النماذج المعقدة ويتطلب فريقاً مدرباً ذا خبرة جيدة في إدارة وتحليل المخاطر ، حيث إن نجاحه يعتمد على تلافي المخاطر في مراحل مبكرة من عمر المشروع ، وإنه في حالة عدم اكتشاف بعض من المخاطر الرئيسية التي تواجه المشروع سوف يؤدي ذلك إلى حدوث مشاكل تعوق تطويره في مراحله الأخيرة ، علاوة على أن هذا النموذج يُعد نموذجاً جديداً لم يتم استخدامه على نطاق واسع في تطوير البرمجيات ، وبالتالي فإن تحديد فعاليته لم تحدد بعد بشكل قطعي ، مما أدي إلى عزوف بعض من العملاء عن استخدامه لأنه من الصعب إقناعهم بأنه يمكن التحكم في تطوير المنتج البرمجي أثناء عملية التطوير.

يُقسم النموذج الحلزوني عملية تطوير البرمجيات إلى عدد من مناطق نشاطات الهيكل **(Framework Activities Regions)** ، بحيث يتم تفعيل كل نشاط هيكلي من خلال تنفيذ العديد من المهام في المنطقة الخاصة به. ففي النموذج الأصلي الذي اقترحه

(Boehm) يوجد أربع مناطق يتم تكييف المهام الخاصة بكل منطقة منها طبقاً لخصائص المشروع الذي يجرى تنفيذه ، وهذه المناطق الأربع ، كما هو موضح بالشكل رقم 3.6 هي كمايلي :

❶ منطقة تحديد الأهداف والبدائل والعوائق : وتشمل المهام التالية :

- تحديد أهداف المنتج في مرحلة التطوير الحالية (الأداء ، والوظيفية ، وسهولة التغيير).
- تحديد العوائق التي تؤثر على تنفيذ مهام المرحلة الحالية (التكلفة ، الموعد النهائي ، التداخل مع مكونات البرنامج الأخرى).
- تحديد الطرق البديلة لتنفيذ المرحلة الحالية (شراؤها ، إعادة استعمال شيء آخر ، تطويرها بأية كيفية).

🕑 منطقة تقويم البدائل وإدارة وتحليل المخاطر وتشمل المهام التالية :

- تقويم خطط التنفيذ البديلة مقارنةً بمجموعة المعايير بواسطة الأهداف والعوائق.
- تحديد مجالات المخاطر.
- تحديد الكيفية التي يمكن بواسطتها التغلب على المخاطر.
- تنفيذ بعض الإجراءات لتقليل المخاطر.

على سبيل المثال ، إذا أعتبر الإيفاء بمتطلبات المستخدم مشكلة محتملة ، ففي هذه الحالة قد يتخذ قرار لتنفيذ بعض النماذج الأولية لتوضيح احتياجات المستخدمين. وفي هذه المرحلة يمكن ممارسة المرونة بدرجة عالية في كل طور من المشروع.

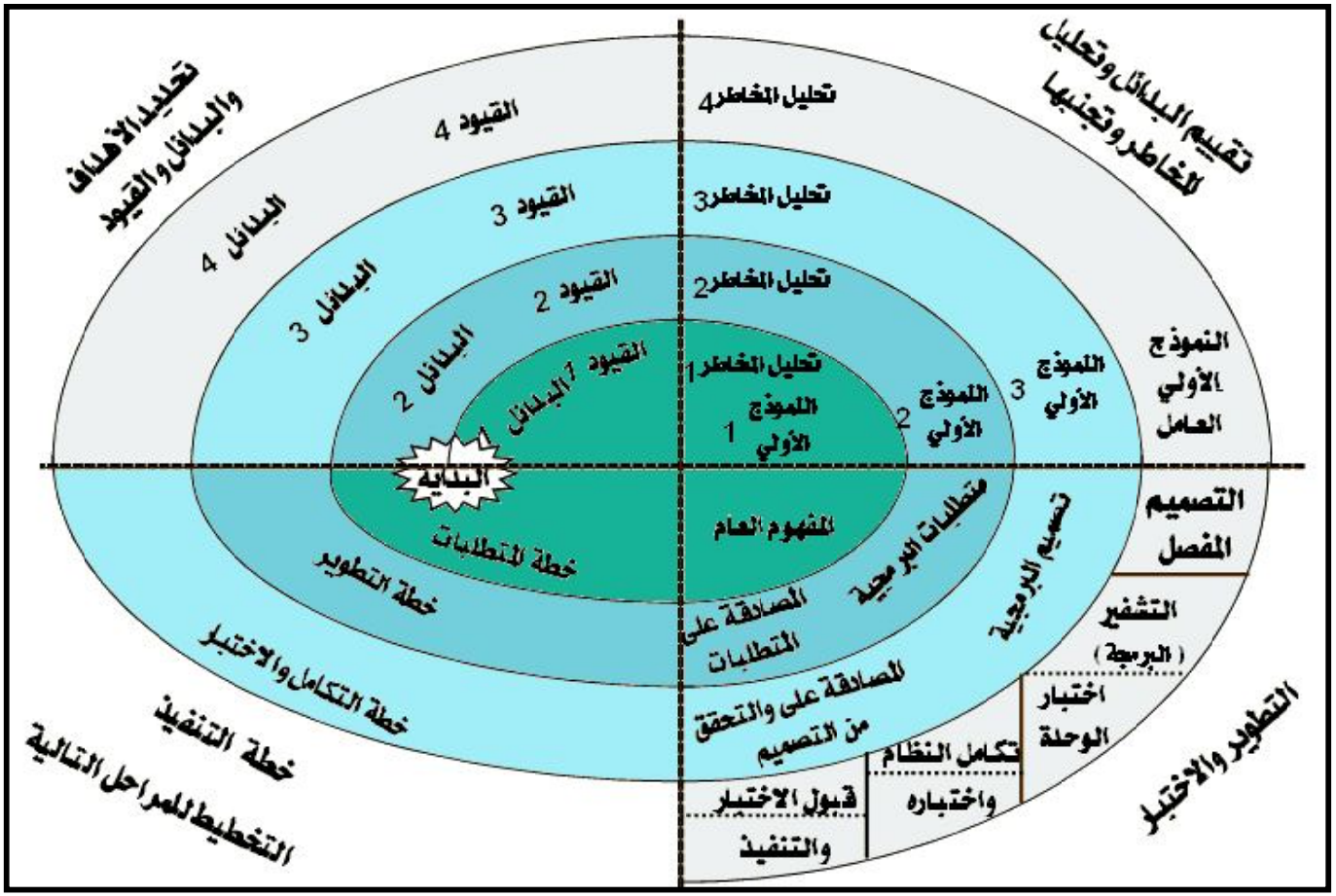
🕑 منطقة التطوير والاختبار : وتشمل المهام التالية :

- تحليل المتطلبات واختيار نماذج لتطوير النظام بناء على تقويم المخاطر ، فمثلاً إذا كانت أكثر المخاطر متعلقة بتصميم وجهة المستخدم ففي هذه الحالة يمكن اختيار نموذج النمذجة الأولية الارتقائي **(Evolutionary Prototyping Model)** لتحديد متطلبات المستخدم ، أما إذا كانت المخاطر متعلقة بعملية تكامل النظم الفرعية للنظام **(Sub-System Integration)** ففي هذه الحالة يفضل استخدام نموذج الشلال

. **(Waterfall Model)**

- تنفيذ المهام الخاصة بعملية الاختبار (التحقق والمصادقة **Verification and Validation)** ، وذلك من خلال الحصول على التغذية العكسية المتمثلة في تقويم العميل للمنتج خلال المرحلة الحالية من التطوير ، فمثلاً إذا كانت المرحلة الحالية تختص بتطوير مواصفات المنتج البرمجي (متطلبات العميل) فتكون التغذية العكسية متمثلة في تضمين أو تعديل أي متطلبات أخرى من العميل يراها ضرورية وذلك بعد تقويمه للمتطلبات التي تم الحصول عليها خلال هذه المرحلة.

🕒 منطقة التخطيط : وتشمل التخطيط للمراحل القادمة من عملية التطوير مثل تعريف الموارد ، وتحديد المواعيد النهائية للمرحلة القادمة من المشروع ، وتحديد عدد الأشخاص الذين سيشاركون في هذه المرحلة ، والتأكد من أن المشروع في المسار الصحيح ، ومعلومات أخرى متعلقة بالمشروع.



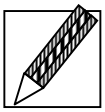
شكل رقم 3.6 : رسم توضيحي للنموذج الحلزوني لتطوير البرمجيات [Pfleeger 98]

وكخلاصة لهذا الموضوع : فإن النموذج الحلزوني هو شبيه بالنموذج التزايدي أو التكراري ، ولكن يتم فيه دمج فعاليات التطوير مع إدارة المخاطر من أجل التحكم فيها وتقليلها، وتبدأ عملية التطوير في النموذج الحلزوني بالدوران حول الحلزون في اتجاه عقرب الساعة من الداخل إلى الخارج ، كما هو موضح بالشكل رقم 3.6. ويبدأ النموذج الحلزوني بمتطلبات العميل مع خطة العمل المبدئية (الميزانية ،و قيود النظام ،و البدائل المتاحة) ، ثم يتقدم خطوة إلى الأمام بتقدير المخاطر وتقويم البدائل المتاحة قبل تقديم ما يُعرف "بوثيقة مفهوم العمليات" **(Concept Of Operation)** ، والتي تصف بشكل عام غير مفصل كيف يجب على النظام أن يعمل. وفي حالة عدم اكتمال تطوير هذا المفهوم ، والذي يُعد منتجاً في حد ذاته ، خلال التزايد الأول (الدورة الأولى على مسار الحلزون الداخلي) تحدث تزايدات متعددة على نفس المسار الحلزوني حتى يكتمل تطوير المفهوم، وبعد ذلك يتم تدقيق محتويات هذه الوثيقة والتأكد من أنها تامة ودقيقة إلى أقصى حد ممكن، وبذلك تكون "وثيقة مفهوم العمليات" هي المنتج من الطور الأول من عملية التطوير، ومن ثم يتم البدء في الدوران حول الحلزون للبدء في الطور الثاني من عملية التطوير والذي يهدف في النهاية إلى إخراج "المتطلبات" كمنتج أساسي له، وفي الطور الثالث تتم عملية "التصميم" ، أما "التنفيذ والاختبار" فتتم في الطور الرابع من عملية التطوير، وفي كل طور من هذه الأطوار يتم المرور على المناطق الأربع وتنفيذ مهام كل منطقة على حده، أي إنه في كل طور يتم تحليل المخاطر والعمل على تقليلها وتقويم البدائل المختلفة لهذا الطور في ضوء متطلبات وقيود النظام ، واستخدام النمذجة الأولية للتحقق من ملائمة أي بديل قبل اعتماده ، وكذلك ضبط خطة المشروع وضبط التكلفة والجدول الزمني بناء على تقويم العميل للمنتج في كل طور وذلك خلال المرور عبر منطقة التخطيط.

نلاحظ في الشكل رقم 3.6 أن الطور الأخير من النموذج الحلزوني (منطقة التطوير والاختبار) مناظر للمراحل الثلاثة الأخيرة من نموذج الشلال (مرحلة التصميم ، ومرحلة التطوير ، ومرحلة تسليم المنتج) مع اختلاف بعض المسميات. حيث إنه عند بداية هذه المنطقة

تكون متطلبات العميل مفهومه وواضحة ومكتملة تماماً لفريق التطوير وذلك من خلال تطـــوير العديد من النماذج الأولية ، بالإضافة إلى تضمين جميع الاقتراحات والحلول في تطوير النموذج الأخير (العامل) لحل جميع المخاطر الرئيسية التي واجهت المشروع في الأطوار السابقة .

تدريب (2)



ارسم شكلاً توضيحياً وبين فيه التناظر بين النموذج الحلزوني ونموذج الشلال.

والسؤال الآن ماذا عن مرحلة الصيانة في النموذج الحلزوني : للإجابة علـــى هـــذا السؤال يجب أن نعرف أن النموذج الحلزوني يمكن تكييفه لتطبيقيه على امتـــداد كامـــل حيـــاة البرمجيات (أي حتى نهاية عمر البرنامج) ، حيث إن عملية الصيانة يمكن اعتبارها طور مـــن أطوار حياة البرمجيات ويتعامل مثل الأطوار الأخرى من حيـــث خضـــوعه لتقـــويم المخـــاطر الناتجة من إجراء عمليات الصيانة المطلوبة لمعرفة إمكانية تنفيذ هذه العمليـــات تحـــت القيـــود المفروضة من عدمه.

أسئلة تقويم ذاتي

ما هي أهم مميزات وعيوب النموذج الحلزوني؟.

"يقسم النموذج الحلزوني عملية تطوير البرمجيات إلى عدة نشاطات هيكليـــة"

اشرح هذه العبارة.

6. نموذج الطرق المنهجية (Formal Methods Model)

ينطلق هذا النموذج من مبدأ : أن تطبيـــق علـــم الرياضـــيات بطرقـــه المنهجيـــة (الصورية) في مجال تطوير البرمجيات يُعد ضرورة في جميع أطـــوار حياتهـــا ، لمـــا تحتويه هذه الطرق من آليات موثوق بها يمكن استخدامها لإزالة العديد من المشاكل التي تواجه عملية تطوير البرمجيات والتي يصعب حلها باستخدام الطرق (النماذج) التقليدية ، حيث إنه يمكن اكتشاف الأخطاء وحالات الغموض الدقيقة التـــي تحـــدث أثنـــاء عمليـــة

التطوير وتصحيحها ، وكذلك ضمان التوافق بين جميع مكونات البرنامج ، عن طريق التحليل الرياضي الدقيق الذي يُمكِّن فريق التطوير من اكتشاف وتصحيح هذه الأخطاء التي يُمكن أن تبقى كامنة طوال فترة التطوير في حالة استخدام الطرق التقليدية التي تعتمد على المراجعة اليدوية أو البصرية أو التطبيقية.

يشمل نموذج الطرق المنهجية (الصورية) مجموعة من الأنشطة التي تُمكِّن فريق التطوير من وصف متطلبات النظام والتحقق من صحة عمله عن طريق تدوين رياضي دقيق يعتمد على استخدام التحليل الرياضي المنهجي (الصوري) **(Formal Mathematical Transformation)** في وصف النماذج بمجموعة من النماذج (المعادلات) الرياضية ، ومن ثم تحويله إلى برنامج تنفيذي.

ويتميز نموذج الطرق المنهجية بما يلي :

- دقة المواصفات المستخلصة عن طريق التحليل الرياضي.
- المحافظة على صحة المعلومات أثناء العمليات التحويلية التي تقود بدورها إلى ثبات المواصفات كمحصلة نهائية .
- يتم تنفيذ جميع عملياتها بطريق آلية.
- إمكانية إعداد الرسوم البيانية المسبقة للمواصفات كدليل على تحقيقها.

ورغم أن تطبيق نموذج الطرق المنهجية يعطي برمجيات دقيقة تتوافق مع جميع متطلبات العميل ، ويُعتمد عليه في حالة تطوير أنظمة تتطلب قدراً كبيراً من الدقة والأمان **(Strict Safety)** ، والاعتمادية **(Reliability)** ، إلا أنه له سلبيات من أهمها :

- أنه مكلف ويستغرق وقتاً طويلاً في عملية التطوير.
- أنه أدوات عملية التحويل المستخدمة فيه نادرة.
- أنه قلة فقط من مطوري البرمجيات قادرون على استخدامه في تطوير البرمجيات.
- أنه يمثل مصدر صعوبة للتواصل مع العملاء حيث إنه يتطلب عميلاً متطوراً عملياً وتقنياً.

أسئلة تقويم ذاتي

1.اسرد أهم المميزات والسلبيات لنموذج الطرق المنهجية.

2. ضع علامة (√) أمام الإجابة الصحيحة وعلامة(X) أمام الإجابة الخطأثم صحح الجمل الخطأ :

أ) نماذج عمليات تطوير البرمجيات هي مجرد إستراتيجية تطوير لتمثيل أنشطة عمليات البرمجيات حيث تؤدي في النهاية إلى إنتاج منتج برمجي قابل للتنفيذ.

ب) لا يتطلب نموذج الشلال تحديد متطلبات العميل بكل دقة ودرجة وضوح كافية قبل البدء في عملية التصميم.

ج) يتم اللجوء إلى نموذج النمذجة الأولية عندما لا تتوافر معطيات كافية محدده من قبل العميل لبناء البرنامج ولكن يعرف مجموعة من الأهداف والخطوط العامة للنظام.

د) استخدام نموذج النمذجة الأولية في تطوير البرمجيات يعطي العميل فكره عامه وجيده عن النظام الفعلي.

هـ) يعتمد نموذج التطوير السريع للتطبيقات على تقسيم المشروع البرمجي إلى عدة وظائف برمجية رئيسية يتم تناول كل منها بواسطة فريق عمل مستقل.

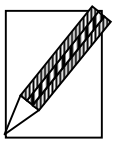
و) تقنيات إعادة الاستخدام (CASE) ، وتقسيم العمل هما حجر الزاوية في نموذج التطوير السريع للتطبيقات.

ز) لا يتطلب نموذج التطوير السريع للتطبيقات وضوح المتطلبات بدرجة عالية.

ح) النموذج التزايدي ملائم في حالة عدم توفر فريق كامل في الشركة الواحدة لإنجاز العمل المناط بها في الموعد المحدد للمشروع.

ط) النموذج الحلزوني في معظم الأحوال يدعم نموذج النمذجة الأولية بطبيعته التكرارية في الإطار النظامي لنموذج الشلال.

تدريب (3)

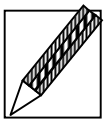


غالباً ما تكون نماذج تطوير عمليـــات البرمجيـــات المختلفـــة متحدة : ناقش هذه العبارة.

نشاط

إن أفضل نموذج لإنشاء نظام معلومات صغير في المكتـــب يجـــب أن يكون مختلفاً عن النموذج المصمم لتطوير نظام يـــتحكم فـــي الأقمـــار الصناعية ، ناقش هذه العبارة.

تدريب (4)



أختر النماذج التي يفضل استخدامها في تطوير عمليات البرمجيات في الحالات التالية (يمكن أن تختار أكثر من نموذج مع فرض توافر باقي مقومات الاستخدام لجميع الأنظمة المذكورة) :

(1) متطلبات العميل واضحة تماماً.

○ نموذج الشلال ○ النموذج الأولي ○ النموذج الحلزوني ○ نموذج **RAD**

(2) متطلبات العميل تتغير أحياناً أثناء دورة التطوير.

○ نموذج الشلال النموذج الأولي ○ النموذج الحلزوني ○ النموذج التزايدي

(3) من الضروري البرهنة على تحقيق المتطلبات المطلوبة أثناء عملية التطوير.

○ نموذج الشلال ○ النموذج الأولي ○ النمـــــــــوذج ○ نموذج **RAD** الحلزوني

(4) عدم كفاية الموارد البشرية ذات الكفاءات المطلوبة لإنجاز المشروع في الوقت المحدد .

○ نموذج الشلال ○ النمـــــوذج ○ النموذج التزايدي ○ نمـــــــوذج الأولي **RAD**

(5) عدم توافر موارد بشرية كافية لتكوين العدد المطلوب من فرق العمل .

○ **نموذج الشلال** ○ **النمـــــوذج** ○ **النموذج التزايدي** ○ **نموذج** RAD **الأولي**

(6) بعض أعضاء فريق التطوير معرضين للاستقالة أثناء عملية التطوير .

○ **نموذج الشلال** ○ **النمـــــوذج** ○ **النمـــــــــوذج** ○ **نموذج** RAD **الأولي** **الحلزوني**

(7) ممثلوا العملاء لن يستطيعوا متابعة عملية التطوير بصورة منتظمة وفعالة .

○ **نموذج الشلال** ○ **النمـــــوذج** ○ **النموذج الحلزوني** ○ **نموذج** RAD **الأولي**

(8) ممثلوا العملاء جديدون بالنسبة لتعريف النظام .

○ **نموذج الشلال** ○ **النموذج الأولي** ○ **النموذج الحلزوني** ○ **نموذج** RAD

(9) ممثلوا العملاء لديهم الرغبة في المشاركة الفعالة في جميع مراحل عملية التطوير.

○ **نموذج الشلال** ○ **النموذج الأولي** ○ **النموذج الحلزوني** ○ **نموذج** RAD

○ **نموذج الشلال** ○ **النموذج التزايدي** ○ **النموذج الحلزوني** ○ **نموذج** RAD

(10) النظام يتطلب قدراً كبيراً من الثقة والأمان والإعتمادية.

○ **نموذج الشلال** ○ **النموذج الأولي** ○ **النموذج الحلزوني** ○ **النموذج المنهجي**

(11) النظام يمثل توجه جديداً للشركة.

○ **نموذج الشلال** ○ **النموذج الأولي** ○ **النموذج الحلزوني** ○ **نموذج** RAD

(12) النظام يمثل تطويراً لنظام موجود بالفعل.

○ **نموذج الشلال** ○ **النموذج التزايدي** ○ **النموذج الحلزوني** ○ **نموذج** RAD

(13) الدعم المالي للمشروع غير مستقر أثناء عملية التطوير.

○ **نموذج الشلال** ○ **النموذج التزايدي** ○ **النموذج الحلزوني** ○ **نموذج** RAD

(14) سوف يتم استخدام النظام لفترة طويلة .

نشاط

بعد انتهاء دراستك للوحدة ارسم جدولاً قارن فيه بين نمـــاذج عمليـــات تطوير البرمجيات من ناحية:

- **وضوح المتطلبات من عدمه. • نوعيات التطبيقات.**
- **الارتباط بالنماذج الأخرى.**
- **الدور الذي يقوم به النموذج للعميل.**
- **المميزات • السلبيات • وقت التطوير**
- **اعتماده على CASE.**
- **توفر الموارد البشرية (العمالة) اللازمة.**

الخلاصة

تناولت هذه الوحدة الأمور التالية:

– نموذج الشلال: الذي يبدأ بمرحلة تجميع المتطلبات بغرض توثيق متطلبات النظام ، ثم مرحلة التصميم وفيها يتم تحويل وثيقة توصيف المتطلبات إلى وثيقة توصيف التصميم ، ثم مرحلة التنفيذ وفيها يتم تحويل وثيقة توصيف التصميم إلى منتج برمجي ، ثم مرحلة الاختبار وتشخيص الأخطاء وذلك للتأكد من خلو البرمجية من الأخطاء ، ثم مرحلة الاختبار والارتقاء وفيها يتم صيانة وتغير بعض من أجزاء النظام البرمجي ، وقد عرفنا أن هذا النموذج يتميز بالبساطة في الفهم والإدارة.

– نموذج النمذجة الأولية: تعتمد فكرة هذا النموذج على التطوير السريع ويتم اللجوء إليه عندما لا تتوافر متطلبات كافية ومحددة عند العميل ، ومن عيوب هذا النموذج أنه بعد الانتهاء من تحديد متطلبات العميل النهائية لا يتم الاستفادة من الإصدارات الأولية التي تم تطويرها لهذا يطلق على هذا النموذج "أسلوب النمذجة الأولية الملقاة جانباً" ، وهناك أسلوب أخر من هذا النموذج في تطوير البرمجيات يسمى "أسلوب النمذجة الأولية الارتقائية".

– نموذج التطوير السريع للتطبيقات وهو يعتمد على تقسيم المشروع البرمجي إلى عدة وظائف رئيسة يتم تناول كلاً منها بوساطة فريق عمل مستقل ، وأهم نشاطات هذا النموذج : عملية النمذجة: (نمذجة الإعمال ،و نمذجة البيانات ،و نمذجة العمليات) وعملية بناء البرمجية (توليد التطبيق ، توليد الشفيرة ، البرمجية آلياً) ، عملية الاختبار (اختبار كامل البرمجية ، تجربة كل الواجهات – الشاشات–) .

– النموذج التزايدي: من النماذج الارتقائية التطويرية ، حيث إنه يجمع عناصر نموذج الشلال التتابعية (التحليل – التطوير – الاختبار) بصورة تكرارية ، وهذا النموذج مفيد في حالة تطوير المشاريع العملاقة.

– النموذج الحلزوني: من النماذج الارتقائية ، في كثير من الأحيان يقوم هــذا النمــوذج بدمج نموذج النمذجة الأولية في الإطار النظامي لنموذج الشلال مع العلم بأن كارثــة أو مصيبة يمكن أن تحدث خلال تطوير المشروع البرمجي ، يقســم النمــوذج الحلزونـــي عملية التطوير إلى عدد من مناطق نشاطات الهيكل وهي : منطقــة تحديــد الأهــداف والبدائل والعوائق ، منطقة تقويم البدائل وإدارة وتحليل المخــاطر ، ومنطقــة التطــوير والاختبار ،و منطقة التخطيط.

نموذج الطرق المنهجية : يشمل مجموعة من الأنشطة التي تمكن فريق التطــوير مــن وصف متطلبات النظام والتحقق من صحة عملة عن طريق تدوين رياضي دقيق يعتمــد على استخدام التحليل الرياضي المنهجي.

لمحة مسبقة عن الوحدة التالية

تعد هندسة متطلبات البرمجيات من أهم الأنشطة والخطوات في بناء وتطوير البرمجيات ، هذا ما سنعرفه في الوحدة التالية وهى بعنوان "هندسة متطلبــات البرمجيــات" وفيهــا سنوضح كل من مفهوم هندسة المتطلبات ، والأنشطة (المراحل) التــي تشــملها عمليــة هندسة المتطلبات ومدخلاتها ومخرجاتها.

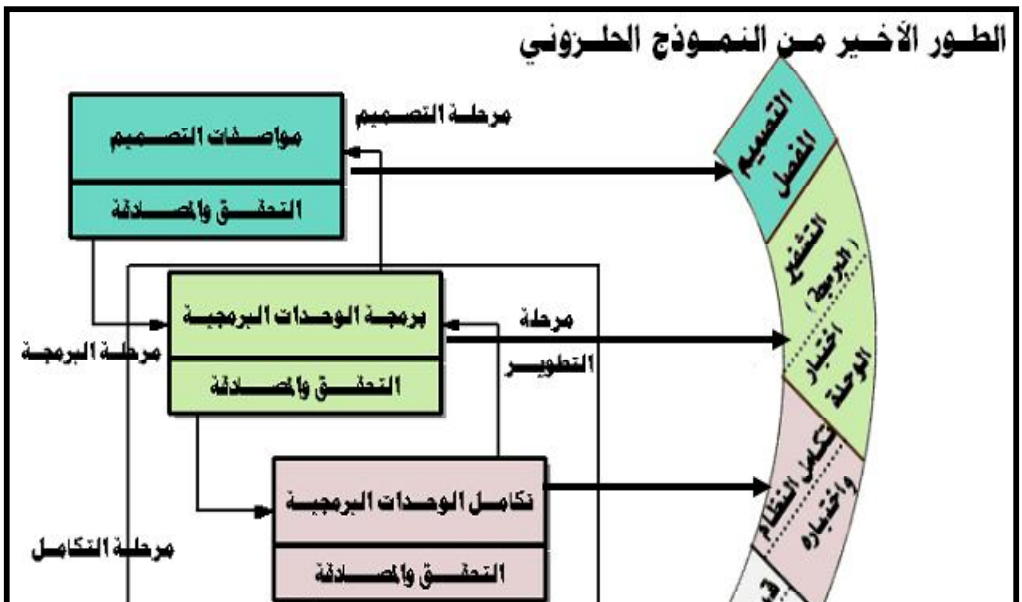
إجابات التدريبات

تدريب (1)

يمكن تلخيص فوائد النموذج فيما يلي:

- تمكن المطورين من التعامل بنجاح في حالة عدم وضوح المتطلبات.
- إتاحة الفرصة للعميل لتغيير رأيه قبل الالتزام النهائي.
- يمكن تحديد متطلبات المستخدم بسهولة.
- تطوير الأنظمة بشكل أسرع.
- تخفيض جهد التطوير لأن النظام الناتج هو النظام الصحيح.
- تخفيض جهد الصيانة لان النظام يلبي احتياجات العميل.
- تسهيل مشاركة العميل الأساسي.
- دعم الاتصال بين العميل والمطور.
- العملاء لا يشعرون بالإحباط أثناء انتظارهم للنظام النهائي لأنهم يشاهدون النظام في حالة تشغيل.
- وجود فرصة سانحة لكي يصبح النظام أكثر صداقةً للمستخدم.
- ستصبح الأنظمة سهلة الفهم والاستعمال بالنسبة للعملاء لأنهم يعرفون ماذا يتوقعون.
- تمكين النظام من التقديم التدريجي للمؤسسة.
- تسهيل تدريب العميل أثناء عملية التطوير.
- هنالك قناعة متزايدة لدى العميل بالبرنامج المستلم.

تدريب (2)



تدريب (3)

من خلال ما سبق يمكن القول إنه يوجد العديد من النماذج المختلفة لتطوير عمليات البرمجيات ، ولكل منها نقاط قوة ونقاط ضعف ، ولكنها تشترك في بعض المراحل العامة لتطوير البرمجيات وعلى الرغم من اختلافها في طرق التنفيذ إلا أن الكثير منها متحد مع الآخر، فمثلاً النموذج الحلزوني يستخدم نموذج النمذجة الأولية عندما تتطلب عملية التطوير ذلك ، وكذلك يمكن أن يَستخدم نموذج الشلال في عملية التطوير في مراحله النهائية، وكذلك بعد أن يتم استخدام نموذج النمذجة الأولية في تحديد المواصفات النهائية للمنتج البرمجي ، باستخدام نماذجه الأولية التي يتم التخلص منها بعد الاستخدام ، يمكن اللجوء إلى نموذج الشلال لإتمام عملية التطوير. وكذلك أثناء

اختبار عملية دمج المكونات البرمجية للمنتج البرمجي يمكن استخدام طريقة التطوير التزايدي بالاشتراك مع أي نموذج آخر.

تدريب (4)

(1) نموذج الشلال ، أو نموذج **RAD**.
(2) النموذج الأولي ، أو النموذج الحلزوني.
(3) النموذج الأولي ، أو النموذج الحلزوني ، أو نموذج **RAD**.
(4) النموذج التزايدي.
(5) نموذج الشلال ، أو النموذج الأولي ، أو النموذج التزايدي.
(6) النموذج الأولي ، أو النموذج الحلزوني.
(7) نموذج الشلال.
(8) النموذج الأولي ، أو النموذج الحلزوني.
(9) النموذج الأولي ، أو نموذج **RAD**.
(10) النموذج المنهجي.
(11) النموذج الأولي ، أو النموذج الحلزوني.
(12) نموذج **RAD**.
(13) النموذج الحلزوني.
(14) نموذج الشلال ، أو النموذج الحلزوني.

مسرد المصطلحات

Risk:

يعرف المصطلح "Risk" على أنه كارثة أو مصيبة يمكن أن تحدث خلال تطوير المشروعات البرمجية.

وثيقة مفهوم العمليات Concept Of Operation:

هي التي تصف بشكل عام غير مفصل كيف يجب على النظام أن يعمل.

المصطلح بالإنجليزية	معناه بالعربية
Application Generation	عملية توليد التطبيق
Automatic Code Generation	توليد الشيفرة البرمجية آلياً
Business Modeling	نمذجة الأعمال
Data Modeling	نمذجة البيانات
Data Structure	بنية البيانات
Documentation	عملية التوثيق
Evolutionary Models	النماذج الارتقائية (التطوُّرية)
Evolutionary Prototyping Model	نموذج النمذجة الأولية الارتقائي
Evolutionary Prototyping Technique	أسلوب النمذجة الأولية الارتقائية
First Prototype Version	الإصدارة الأولى الأولية
Formal Mathematical Transformation)	التحليل الرياضي المنهجي (الصوري)
Formal Methods Model	نموذج الطرق المنهجية
Fourth Generation Technology	تكنولوجيا الجيل الرابع
Framework Activities Regions	مناطق نشاطات الهيكل
Fully Functional Style	كنظام يعمل تماماً
Incremental Mode	النموذج التزايدي
Interface Design	تصميم وجهة الاستخدام
Intermediate Prototype Versions	الإصدارة الأولية المعدلة
Modeling	عملية النمذجة

المصطلح بالإنجليزية	معناه بالعربية
Oracle Developer and Designer	مطوِّر ومصمم أوراكل
Outline Specification	الخطوط العامة للبرنامج
Processes Modeling	نمذجة العمليات
Product Core	نواة المنتج
Prototyping Model	نموذج النمذجة الأولية
Rapid Application Development "RAD" Model	نموذج التطوير السريع للتطبيقات
Rapid Development	التطوير السريع
Reliability	الاعتمادية
Software Architecture	بنيان البرمجية
Software Development Processes Models	نماذج عمليات تطوير البرمجيات
Strict Safety	الدقة والأمان
Sub-System Integration	تكامل النظم الفرعية للنظام
Systematic Framework	الإطار النظامي
Technological Breakthroughs	ظهور تقنية جديدة
Throw-Away Prototyping Technique	أسلوب النمذجة الأولية الملقاة جانبا
Uncertainty	أشياء مجهولة أو مشكوك في صحتها
User Interface	واجهة الربط البنية مع المستخدم
Verification and Validation	التحقق والمصادقة
Waterfall Model	نموذج الشلال
Waterfall Model	نموذج الشلال
Work and Data flow	تدفق الأعمال والبيانات

المراجع

أولاً : المراجع العربية :

[1] روجر بريسمان ، "هندسة البرمجيات" ، الدار العربية للعلـــوم ، مركـــز التعريـــب والبرمجة ، الطبعة الأولى ، 1425هـــ – 2004م .

http://www.w3.org

[2] أسماء المنقوش ، "دورة هندسة البرمجيات" ، منشور على الموقع :

http://www.c4arab.com/

المرجع الأساسي لهذه الدورة هو :

Shari Pfleeger, "Software Engineering - Theory and Practice", 2nd Edition

ثانياً : المراجع الإنجليزية :

[3] Ian Somerville , "Software Engineering", Addison Wesley, 2001.

[4] Ronald J. Leach,, "Introduction to Software Engineering", CRC Press, 1999.

[5] Douglas Bell , "Software Engineering A Programming Approach", 3rd Edition, Addison Wesley.

[6] **Shari Pfleeger, "Software Engineering - Theory and Practice", 2nd Edition.**

[7] Boehm, B. (1988), "A Spiral Model of Software Development and Enhancement", IEEE Computer 21, 5, 61-72.

[8] Royce, W. (1970), "Managing the Development of Large Software Systems : Concept and Techniques", Proc. WESCON, August 1970.

ثالثاً : مواقع على شبكة الإنترنت تم الاستفادة منها :

[9] http://forum.amrkhaled.net/

[10] http://ar.wikipedia.org/wiki
[11] http://www.uop.edu.jo/Arabic/Faculties/

[12] http://cs.wwc.edu/

[13] http://www.pcwebopedia.com/

[14] http://courses.cs.vt.edu/~csonline/SE/Lessons/

[15] http://scitec.uwichill.edu.bb/cmp/online/

[16] www.bit.lk/information/tv/s2/2003/IT2401c.ppt

[17] http://www.cs.mu.oz.au/341/Lectures

[18] http://www.arabteam2000-forum.com

[19] www.cise.ufl.edu/class/cen5035/lecture_notes.html

[20] www.qucis.queensu.ca/Software-Engineering/

[21] www.c4arab.com/


الوحدة الرابعة
هندسة متطلبات
البرمجيات

محتويات الوحدة

المقدمة

تمهيد

عزيزي الدارس ،،

أهلاً بك في الوحدة الرابعة من مقرر " هندسة البرمجيات(1)" ، هذه الوحدة تتناول هندسة متطلبات البرمجيات والتي سنتعرف فيها على:

– مفهوم هندسة البرمجيات ، وسوف نركز هنا على بعض الأنشطة المهمة التي تشملها عملية هندسة المتطلبات وهي : استنباط المتطلبات ، والتفاوض حول المتطلبات وتحليلها ،و توثيق المتطلبات ،و المصادقة على المتطلبات.

– مفهوم المتطلبات: وهنا سنلقي الضوء على الفرق بين المتطلبات والتصميم ، وكذلك الفرق بين متطلبات النظام ومتطلبات البرمجيات ، وكذلك الفرق بين متطلبات العميل ومتطلبات المطور.

– أنواع المتطلبات وهي تقسم إلى نوعين رئيسين هما : متطلبات وظيفية (وصف للخدمات التي يجب أن تقدمها البرمجية بالتفصيل) ، متطلبات غير وظيفية (وصف للقيود المفروضة على الخدمات التي يجب أن يقدمها النظام.)

– وثيقة مواصفات المتطلبات: أو المنتج النهائي لعملية هندسة المتطلبات ، وهي ضرورية لنجاح وتطوير أي مشروع تطوير برمجية ، وسوف نتناول في هذا الجزء مستويات المتطلبات ،و بنية مستند مواصفات المتطلبات.

عزيزي الدارس أرجو أن لا يغيب عن ذهنك عند قراءة مادة الوحدة محاولة الإجابة عن الأسئلة التقويمية ، والعودة إلى النص للتأكد من صحة الإجابة ، كما أرجو عدم الرجوع إلى إجابات التدريبات إلا بعد محاولة حلها أولاً.

نأمل أن تؤدي دراستك إلى هذه الوحدة إلى إثراء فهمك وتعرفك هندسة متطلبات البرمجيات.

أهداف الوحدة

عزيزى الدارس بنهاية دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

تشرح مفهوم هندسة المتطلبات.

تصف هيكلاً عاماً لأنشطة (مراحل) عملية هندسة البرمجيات.

تحدد مدخلات ومخرجات عملية هندسة البرمجيات.

تعرِّف مفهوم المتطلبات.

تعدد المستفيدين من تحديد المتطلبات.

تحدد الخصائص المميزة للمتطلبات الجيدة.

تخبر عن الفوائد الناتجة من تحديد متطلبات جيدة.

تشرح مهارات وأساليب الاتصال والإرشادات التي يجب مراعاتها عند استنباط المتطلبات.

تصف دور المحلل والمهارات والأساليب والإرشادات التي يجب مراعاتها عند تحليل المتطلبات.

تختار الأساليب التي يجب استخدامها في عملية المصادقة على المتطلبات.

تعدد المشاكل التي قد تنشأ في مراحل التصميم والتنفيذ والصيانة في حالة وجود خلل في المتطلبات.

تدون العوامل التي يجب مراعاتها عند كتابة وثيقة مواصفات المتطلبات.

تميز بين أنواع المتطلبات المختلفة ؛ الوظيفية منها وغير الوظيفية.

تبوب المحتويات الأساسية (بنية) لمستند مواصفات المتطلبات.

تكتب مستند مواصفات المتطلبات بناءاً على معيار المنظمة العالميةIEEE IEEE.

توطئه

تُعد البرمجيات

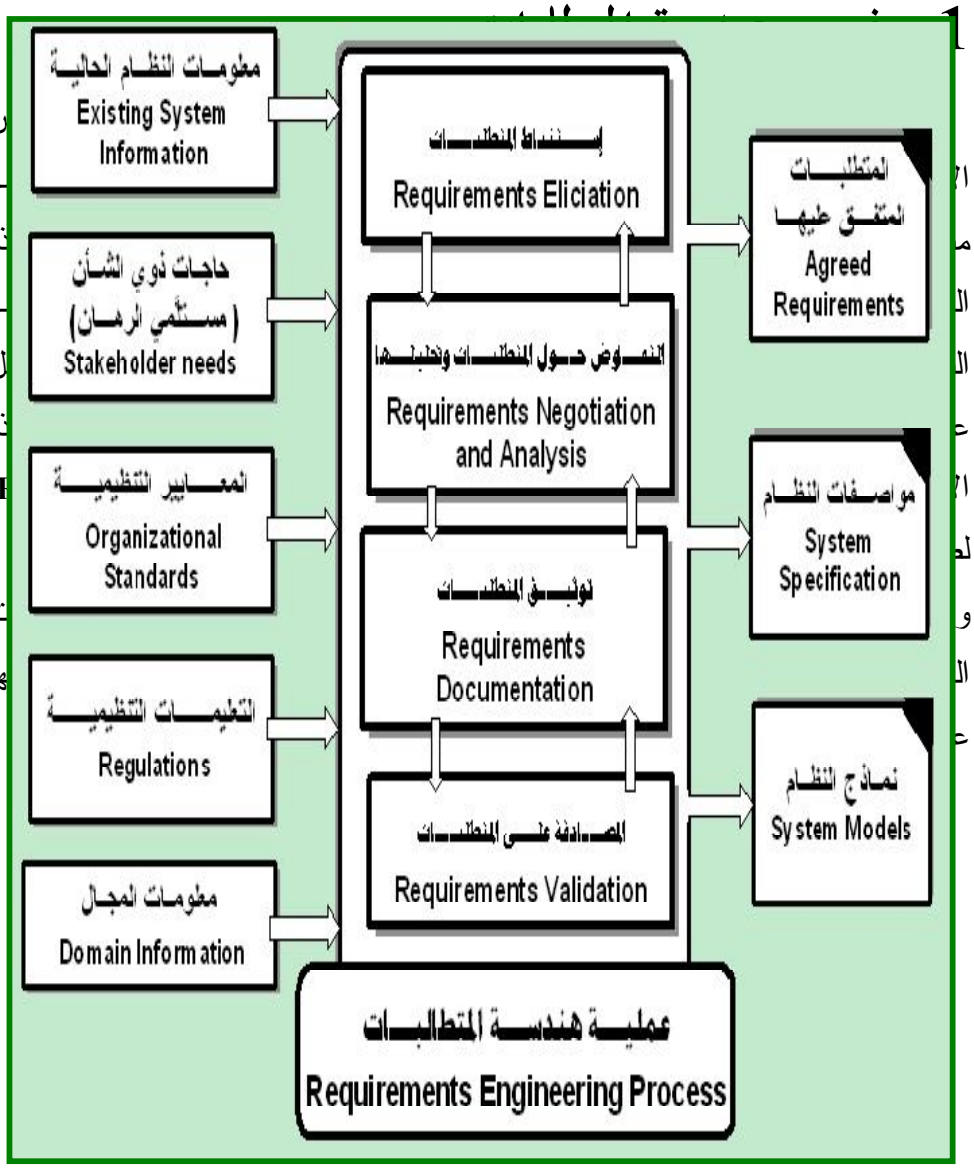
وفي حالة قيامنا ببناء اقوى البرمجيات ، ولكنها لا تفي باحتياجاتنا فإننا نكون قد

فشلنا، ففي العالم الحقيقي لتطوير البرمجيات ، هنالك مؤشرات قوية على أن العديد من النظم لا تفي بمتطلبات المستخدمين وذلك لعدم تحديد متطلبات هذه الأنظمة على وجه الدقة من العملاء اعتقادا منهم بأن متطلبات الأعمال عالية المستوى كافية لتحقيق الأهداف المنشودة من النظم المطلوب تطويرها ، علاوة على صعوبة إقناع العملاء بقضاء بعض الوقت لتغطية تفاصيل المتطلبات.

لذا فإن هندسة متطلبات البرمجيات تُعد النشاط التقني الأول في تطوير البرمجيات في محاولة الضمان والتأكيد على ما يرغبه ويطمح إليه العميل، ويجب أن تستمر هذه الجهود طيلة عملية تطوير النظام والتي يطلق عليها "المصادقة" **(Validation)**. كذلك فإن لتوصيف المتطلبات دوراً حيوياً هاما ثانيا ، حيث إنه المحور الذي يتم به التحقق من أن البرمجيات تعمل بصورة صحيحة وتفي بالغرض المستهدف منه أم لا، كما أن بذل الجهد لضمان حدوث الحد الأدنى من الأخطاء في البرمجيات يحتاج لوقت هائل وعملية صعبة تحدث أثناء عملية التطوير، وتسمى هذه العملية "التحقق" **(Verification)**.

وتنطلق فكرة تطوير نظام معين عندما يكون لدى المستخدم فكرة لنظام جديد (أو تطوير نظام موجود فعلا) ، فإنه يقوم باستدعاء المحلل ومن ثم يتعاونون معاً في وضع التفاصيل المتعلقة بمتطلبات هذا النظام ومواصفاته، وقد تكون الفكرة الجوهرية الأولية لدى المستخدم غامضة وغير محددة بدقة وأحيانا تكون واضحة المعالم.

والسمة الأخرى لتوصيف المتطلبات هي أنه من الضروري وضع الدليل الإرشادي حول كيفيه التأكد من أن البرنامج أو النظام يلبي احتياجات المستخدمين ، وذلك من خلال استخدام أدوات القياس والاختبارات المختلفة.



شكل 4.1 : شكل توضيحي لعملية هندسة البرمجيات : أهم الأنشطة الهيكلية (المراحل) والمدخلات والمخرجات.

وسوف نركز هنا فقط على بعض الأنشطة المهمة التي تشملها عملية هندسة المتطلبات ، وهي كالتالي :

- **استنباط المتطلبات (Requirements Elicitation) .**
- **التفاوض حول المتطلبات وتحليلها**

(Requirements Negotiation and Analysis).

- **توثيق المتطلبات (Requirements Documentation)** .
- **المصادقة على المتطلبات (Requirements Validation)** .

1.1 مراحل (أنشطة) هندسة متطلبات البرمجيات

① **المرحلة الأولى : استنباط المتطلبات :** وهي مسئولية كل من المحللين والمستخدمين بالحوار مع بعضهم البعض بحيث يحاول كل منهم أن يتفهم الأخر، وتؤدي هذه المرحلة إلى بيان غير رسمي للمتطلبات **(Informal Statement of Requirements)**. وهذا يتطلب شكلاً من أشكال الاتصال المباشر والمهارات التي يجب أن يمارسها المحلل ليست تلك المهارات التقنية العادية التي تتعلق بتطوير البرمجيات ، ولكن تشمل العديد من جوانب مهارات الاتصال الأخرى ، حيث تشمل هذه المهارات الاستماع إلى احتياجات المستخدمين وطرح عدة أسئلة تساعدهم على تحديد أهدافهم وعوائقهم ، وفي النهاية تسجيل وجهات نظرهم لمتطلبات النظام. ومن أهم أسئلة استنباط المتطلبات ما يلي :

- ما هي الأسباب وراء تطوير النظام؟ .
- ما هي توقّعات المستخدمين من النظام؟ .
- مَنْ هم المستخدمون وما هو مستوى خبراتهم العملية؟ .
- ما هي مرئيات المستخدمين حول كيفية استخدامهم للنظام؟ .
- ما هي خصائص البيئة التي يجب أن يستجيب لها النظام؟ .
- ما هي واجهات الاستخدام المتاحة والتي سيتم تصميمها؟ .
- ما هي الوظائف التي سيؤديها النظام من وجهة نظر المستخدمين وبلغتهم العامة؟ .
- ما هي القيود و الأجهزة ، والبرمجيات ،و النواحي الاقتصادية والإجرائية التي يجب أن يستجيب إليها النظام؟ .

- ماذا سَيَكُونُ الشكل النهائي للنظام ، شكل النماذج الأولية **(Prototypes)** ، وشكل النظام النهائي الذي سوف يتم على أساسه الإنتاج الشامل ذو الكميات الكبيرة **(Mass Production)** .

ومن أهم الإرشادات التي يجب مراعاتها في هذه المرحلة ، ما يلي :

- الأخذ في الاعتبار السياسات والتنظيمات الداخلية للمؤسسة في عملية تطوير البرمجيات ، وكذلك الاهتمامات المهنية لها.
- استشارة كل من له علاقة بعملية تطوير البرمجيات **(System Stakeholders)**.
- تسجيل منابع المتطلبات **(Requirements Sources)**.
- تعريف بيئة تشغيل النظام **(System's Operating Environment)**.
- دراسة القيود المفروضة من قبل نطاق النظام **(System Domain Constraints)** .
- تسجيل الأسباب المنطقية للمتطلبات **(Requirements Rationale)** .
- تجميع المتطلبات من وجهات نظر متعددة.
- استخدام أساليب / أنشطة **(Techniques / Activities)** مختلفة لاستنباط المتطلبات ، والتي من أهمها :

- **أسلوب المقابلات (Interviews Technique)** : وهي عملية اتصال مباشر بين المستخدمين ومن لهم صله مباشرة بالنظام **(Stakeholders)** ومصممي أو مهندسي البرمجيات، ويطلق عادة على المهندسين الذين يتعاملون مع الاحتياجات بمهندسي المتطلبات أو بمحللي النظم ، وسوف نستخدم في هذه الوحدة مصطلح "محلل" . أما بالنسبة للمستخدمين فإنهم يُعرفون أحياناً بالعملاء أو بالزبائن، وبالطبع هنالك فارق ما بين المستخدم والعميل ، فمثلاً تقوم منشاة أو منظمة (العميل) بشراء نظام معلومات لموظفيها (المستخدمين المعنيين) كي يستخدمونه في أعمالهم ، ومن ثم سوف نستخدم في هذه الوحدة مصطلح "المستخدم" . ويتم خلال هذا الاتصال مناقشة

المتطلبات على هيئة سؤال وجواب ، ويستخدم أسلوب المقابلات في استنباط المتطلبات أو التحقق منها.

- **أسلوب استخلاص البيانات (Data Mining Technique)** : وهو عبارة عن نشاط بحثي خلال أوعية البيانات المختلفة (وثائق **(Documents)** ،و نماذج **(Forms)** ،و قواعد بيانات **(Database)** ، وملاحظات بريد إلكتروني **(E-Mail Notes)** ، ....وخلافه.) بغرض الحصول على معلومات تفصيلية استخراجها وعلى أساسها يتم تحديد متطلبات النظام المطلوب تطويره.
- **أسلوب المحلل كمتدرب (Analyst as Apprentice)** : في هذا الأسلوب يتم مشاركة المحلل مع المستخدمين النهائيين للنظام باعتباره متدرباً للقيام بأعمالهم بغرض إعطاء المحلل فكرة مفصلة ودقيقة عن المهام الخاصة التي يقوم بها المستخدم النهائي للنظام ، والذي هو في النهاية سوف يستخدم النظام المطلوب تطويره.
- **أسلوب التحليل الاجتماعي والمشاهدة (Observation and Social Analysis)** : وهو أسلوب مشابه لأسلوب "المحلل كمتدرب" ، ولكن المحلل يجلس هنا مع المستخدم النهائي لمشاهدته أثناء أدائه لعمله الحالي أو أثناء عمله على نسخة قديمة من النظام المراد تطويره، والغرض من هذا الأسلوب هو اكتساب معلومات تفصيلية عن آلية العمل التي يستخدمها المستخدم النهائي مع تحديد أو معرفة أية آليات ذات كفاءة تم تطويرها واستخدامها من قبله والتي تختلف مواصفات المهام الخاصة به.

②المرحلة الثانية : التفاوض حول المتطلبات و تحليلها : وهي مرحلة يتم فيها تصنيف المتطلبات وتنظيمها في مجموعات واكتشاف العلاقات بينها و بين المتطلبات الأخرى ، حيث يقوم المحلل بتبسيط الأفكار حول المعلومات المعطاة أثناء المرحلة الأولى، وهو يحوّل أفكار المستخدمين حول النظام إلى تقديم تمثيل منظم للنظام كما يراه. وهذا ربما يسبب بعض التعقيدات (ربما يكون هناك عدد من المستخدمين المختلفين لديهم آراء مختلفة عن احتياجاتهم). حيث إنه في مستهل كل مشروع يوجد

على الأقل رأيان هما وجهة نظر مستخدم ووجهة نظر مطـور. وربمـا يكـون هنـاك اختلاف في الرأي بين العملاء ومستخدميهم ، فمثلاً سيهتم المستخدمون بأن يكون النظـام مريحاً لهم وذا إمكانات عالية ــ ربما مبالغ فيها ــ والتي يمكـن أن تـؤدي إلـى زيـادة التكلفة ، وربما سيرفضون أي تسهيلات في النظام لمراقبة معدلات أعمالهم وهو مـا قـد يرفضه العميل نفسه وربما ــ أيضاً ــ لا يصل المستخدمون إلى رأي متجانس فمنهم مـا يعتقد أن الحاسب الآلي يمكن عمل أي شيء فتكون مطالبهم عالية ، وآخرون بسطاء فـي الاتجاه المعاكس يؤمنون بأن الحاسب الآلي يمكن أن ينفذ فقط معظـم الأعمـال العاديـة بصورة آلية ومن هنا يبرز دور المحلل باعتباره مفاوضاً لحل هذه الاختلافات للحصـول على الموافقة من كل المستخدمين والعملاء على مواصفات متطلبات نهائية. وتـؤدي هـذه المرحلة إلى المتطلبات المتفق عليها **(Agreed Requirements)** من مختلـف الأشـخاص والجهات ذات العلاقة بتطوير النظام **(Stakeholders)**.

وبصفة عامة ، يمكن تلخيص دور المحلل في هذه المرحلة في النقاط التالية:

- المساعدة على حل الاختلافات المحتملة في الآراء بين المسـتخدمين أنفسـهم وبينهم وبين العملاء.
- إسداء النصيحة للمستخدمين عما هو ممكن تنفيذه تقنياً وما هو غير ممكن.
- التفاوض مع المستخدمين والعملاء والحصول على مواصفات متطلبات نهائية.

ومن أهم الإرشادات التي يجب مراعاتها في هذه المرحلة ، ما يلي :

- تعريف حدود النظام **(System Boundaries)**.
- التخطيط للصعوبات المتوقعة وكذلك لطرق حلها.
- استخدام أسـاليب و أنشطة **(Techniques / Activities)** مختلفـة لإجـراء عمليـات التفاوض والتحليل ، والتي من أهمها :

- **قوائم تدقيق المتطلبات (Requirements Checklists)** : وتستخدم للتأكـد مـن عدم نسيان أي من المتطلبات في جميع مراحل هندسة المتطلبات ، وكذلك التأكد

من وضوح المتطلبات وقابليتها للاختبار وعدم وجود أية تعارضات أو تكرارات أو مرادفات بين المتطلبات.

- **تعيين أولوية المتطلبات (Requirements Prioritization)** : وهي تنطلق من مبدأ أنه من الجائز عدم تحقيق كامل متطلبات الـ "**Stakeholders**" عند بناء النظام ، ويجب تحديد أوليات تنفيذ المتطلبات من قِبل الـ "**Stakeholders**" حسب الأهمية ، وذلك لتسهيل عملية اختيار المتطلبات الضرورية في حالة وجود مقايضة **(Trade-Off)** بين ما ينجز من متطلبات وبين القيود المفروضة على النظام مثل التكلفة ومواعيد التسليم النهائية.
- **النمذجة غير الاصطلاحية (Informal Modeling)** :وهي تتضمن العديد من الوسائل مثل الوصف النصي **(Text Description)** ، والصور الغنية بالمعلومات **(Rich Pictures)** لوصف النظام بطريقة واضحة ومرنة ومفهومه بحيث تُمكن الـ "**Stakeholders**" من استكشاف خبايا النظام والتحاور فيما بينهم وكذلك فيما بينهم وبين فريق التطوير بدون الحاجة إلى معرفتهم بأساليب النمذجة المختلفة.
- **النمذجة شبه الاصطلاحية (Semi-Formal Modeling)** : بالإضافة إلى النمذجة غير الاصطلاحية تُعد النمذجة شبه الاصطلاحية من أهم الأنشطة في عملية توثيق المتطلبات حيث تتضمن العديد من الوسائل التي تستخدم لتدوين البيانات باستخدام مجموعة من الرموز (الاصطلاحات) **(Notations)** ذات دلالات ومفاهيم معينة مثل مخططات سريان البيانات **(Data Flow Diagrams "DFD")** ،و خرائط الحالة **(State Charts)** ،و لغة النمذجة الموحدة **(Unified Modeling Language "UML")** . والهدف من النمذجة شبه الاصطلاحية هو نفس الهدف من النمذجة غير الاصطلاحية ألا وهو تطوير بيان يصف النظام بطريقة واضحة ومرنة ومفهومه لتسهيل عمليات الاتصال والتحليل ومساعدة المحلل

والمهتمين بالنظام "**Stakeholders**" على فهم النظام ، ولكنها تختلف في الوسائل التي تستخدم في عمليات التدوين حيث إنها تستخدم مجموعة من الرموز المعَرفة سابقاً والتي لها دلالات ومفاهيم وقواعد معينة لوصف النظام.

- **عملية التدرج التحليلي (Analytic Hierarchy Process)** : وهي عملية تساعد عند اتخاذ قرار اختيار ما بين المتطلبات ، حيث تتم عملية الاختيار المعقدة من خلال تقسيمها إلى مجموعة من المقارنات البسيطة بين المتطلبات المراد الاختيار فيما بينها وتستخدم عملية التدرج التحليلي في حالة صعوبة تحديد أوليات تنفيذ المتطلبات أو في حالة اختلاف والمهتمين بالنظام "**Stakeholders**" في اختيار المتطلبات الضرورية.

③**المرحلة الثالثة : توثيق المتطلبات:** وهي كتابة بيان واضح غالباً بلغة طبيعية عما يتوقع أن يقدمه النظام للمستخدمين ، وتؤدي هذه المرحلة إلى مسودة الوثيقة النهائية للمتطلبات **(Draft Requirements Document)** .

ومن أهم الإرشادات التي يجب مراعاتها في هذه المرحلة (بدون ترتيب) ، ما يلي :

- تعريف الهيكل القياسي المستخدم في عملية التوثيق.
- شرح كيفية استخدام الوثيقة.
- تضمين ملخص عن المتطلبات.
- تعريف المصطلحات المستخدمة خلال الوثيقة.
- تصميم الوثيقة بحيث تكون مقروءة للجميع ، وتساعد القارئ على إيجاد المعلومة ، وسهلة التعديل.
- استخدام لغة بسيطة وموجزة.
- استخدام الرسوم التوضيحية على نحو ملائم وواضح.
- وصف المتطلبات بصورة كمية **(Quantitatively)** إن أمكن ذلك.

- استخدام أساليب / أنشطة (**Techniques / Activities**) مختلفة لإجراء عمليات التوثيق والتوصيف ، والتي من أهمها : **النمذجة غير الاصطلاحية (Informal Modeling) ، والنمذجة شبه الاصطلاحية (Semi-Formal Modeling)** ، بالإضافة إلى

- **النمذجة الاصطلاحية (المنهجية) (Formal Modeling)** :وهي تشتمل مجموعة من الأنشطة التي تُمكِّن فريق التطوير من وصف متطلبات النظام والتحقق من صحة عمله عن طريق تدوين رياضي دقيق يعتمد على استخدام التحليل الرياضي المنهجي (الصوري) **(Formal Mathematical Transformation)** ، وتتم تنفيذ جميع عملياتها بطريق آلية ، مع إمكانية إعداد الرسوم البيانية المسبقة للمواصفات كدليل على تحقيقها.،وفي حين تهدف النمذجة غير الاصطلاحية والنمذجة شبه الاصطلاحية إلى تطوير بيان يصف النظام لتسهيل عمليات الاتصال والتحليل ومساعدة المحلل والمهتمين بالنظام "**Stakeholders**" على فهم النظام ، نجد أن هدف النمذجة الاصطلاحية هو تطوير بيان وصفي للنظام يساعد على التحقق المطلق من سلوك وخصائص النظام الضرورية.
- **تتابعية المتطلبات (Requirements Traceability)** : تشتمل على مجموعة من الروابط بين متطلبات النظام وأنواع أخرى من المعلومات والتي من أهمها : مصدر المتطلب **(Requirement Source)** ،و أسباب المتطلب **(The Rationale Behind the Requirement** ، وخلافه.
- **إدارة تغيير المتطلبات (Requirements Change Management)** : وهي العملية التي من خلالها يتم مراجعة التغييرات المقترحة في المتطلبات والتأكد من قبولها من الـ "**Stakeholders**" قبل تضمين هذه التغييرات في وثيقة توصيف المتطلبات ، وبالتالي قبل تنفيذها في النظام. وإدارة تغيير المتطلبات تتضمن أيضاً إدارة التغييرات في النظام وأخطاء النظام الناتجة من التغييرات في المتطلبات.
- **إعادة استخدام المتطلبات (Requirements Reuse)** : وتعني إعادة استخدام مواصفات المتطلبات (أو جزءٍ منها) التي سبق تطويرها في مشاريع سابقة ،

مثل : إعادة استخدام القوالب **(Templates)** الخاصة بالمواصفات العامة ، أو استخدام مواصفات معينة سبق تطويرها لمشاريع مماثلة.

④**المرحلة الرابعة المصادقة على المتطلبات :** وهي مسئولية مختلف الأشخاص والجهات ذات العلاقة بتطوير النظام **(Stakeholders)**. وتؤدي هذه المرحلة إلى إنتاج تقرير عن المصادقة عن المتطلبات **(Requirements Validation Report)** ، ويُعد هذا التقرير حجر الزاوية في عملية هندسة المتطلبات ، حيث إنه بناءً على هذا التقرير أما أن تُقبل مسودة الوثيقة النهائية للمتطلبات المنتجة في المرحلة السابقة وتصبح وثيقة نهائية ، أو يتم إعادة تكرار العملية مرة أخرى ، كما هو موضح بالشكل رقم 4.1. وسوف نتناول التحقق والمصادقة بصفة عامة ــ إن شاء الله ــ في الوحدة القادمة ، ولكن سوف نذكر هنا ــ لجعل الموضوع متكاملاً ــ بعض من الأساليب و الأنشطة **(Techniques / Activities)** التي يتم استخدامها لإجراء عمليات المصادقة على المتطلبات ، والتي من أهمها هما **النمذجة الأولية (Prototyping) بنوعيها : النمذجة الأولية الملقاة جانباً (Throw-Away Prototyping) ، والنمذجة الأولية الارتقائية (Evolutionary Prototyping)** ، وقد تم مناقشتهما في الوحدة السابقة ، وكذلك استخدام **قوائم تدقيق المتطلبات (Requirements Checklists)** والتي تم شرحها سابقاً خلال هذه الوحدة ، هذا بالإضافة إلى :

- **فحص المتطلبات (Requirements Reviews)** : وتشتمل هذه العملية على فحص الـ "**Stakeholders**" مواصفات المتطلبات للتأكد من أنها تفي بجميع متطلباتهم ، ومن ثم الاجتماع مع مهندسي فريق التطوير لإطلاعهم على نتائج الفحص والتي تُعد نوعاً من التغذية العكسية **(Feedback)** التي تساعد فريق التطوير على تلافي القصور الموجود في مواصفات المتطلبات ، وتنتهي هذه العملية بعد موافقة الـ "**Stakeholders**" على مواصفات المتطلبات المقدمة إليهم.

أسئلة تقويم ذاتي

عدد أهم الأسئلة التي تساعد في استنباط المتطلبات ، وما هي أهم الإرشادات التي يجب مراعاتها في هذه المرحلة.

2.1 مدخلات ومخرجات عملية هندسة متطلبات البرمجيات

لإتمام عملية هندسة المتطلبات بطريقة تضمن استنتاج متطلبات تفي وبصورة كاملة باحتياجات كل من المستخدم والعميل يجب توافر بعض المدخلات الضرورية والموضحة بالشكل رقم 4.1 ، وهي كالتالي :

- **معلومات النظام الحالية (Existing System Information)** : وهي معلومات عن الأداء الوظيفي للنظام المطلوب توصيف متطلباته ، وكذلك الأنظمة الأخرى التي يتفاعل معها هذا النظام.
- **حاجات ذوي الشأن (أصحاب المصلحة) (Stakeholder Needs)** : وهي تمثل وصف متطلبات ذوي الشأن أو أصحاب المصلحة **(Stakeholder)** من تطوير النظام لدعم الأعمال القائمين بها، وبصفة عامة يُعرف **ذوي الشأن أو أصحاب المصلحة (Stakeholder)** بأنهم أفراد أو مجموعات معينة لها تأثير هام أو يمكن أن تتأثر أو لها مصلحة مباشرة بـ "مجموعة من الأنشطة" المتعلقة بتطوير منتج ما ، أي أنهم هم الأشخاص المهتمين بإنجاز عملية تطوير المنتج ، ويتم تعيينهم بشكل طبيعي بناءًعلى أدوارهم أو وظائفهم وليس بصفتهم الشخصية ، ومن أهم هؤلاء : مهندسوا البرمجيات **(Software Engineers)** المسئولين عن تطوير النظام ،و المستخدمون النهائيين للنظام **(System End-Users)** الذين سوف يستخدمون النظام بعد تسليمه ، والمدراء المسئولون عن المستخدمين النهائيين للنظام **(Managers of System End-Users)** ، والمنظمون الخارجون **(External Regulators)** الذين يتأكدون من مطابقة النظام المطور للمواصفات الصحيحة ، و الخبراء (Experts) الذين يعطون المعلومات الأساسية عن نطاق تطبيق النظام.

- **المعايير التنظيمية (Organizational Standards)** : تستنتج هذه المتطلبات من السياسات والإجراءات التي تتبعها مؤسسة العميل في تطوير البرمجيات ، مثل المعايير القياسية المستخدمة في المنظمة والمعنية بتنفيذ وتطوير الأنظمة وإدارة جودتها (من أهم المعايير المستخدمة في مثل هذه الحالات معيار وتعليمات IEEE ،و المواصفات والمعايير العسكرية ، ومعايير ISO) ، ومتطلبات التنفيذ (مثل نوع اللغة البرمجية التي سوف تستخدم) ، وطريقة التصميم والبناء الهيكلي ، ومتطلبات التسليم.
- **التعليمات التنظيمية (Regulations)** : وهي تمثل التعليمات والقيود المفروضة على النظام مثل تعليمات الصحة ، والأمن التي تطبق على النظام.
- **معلومات المجال (Domain Information)** : تستنتج هذه المعلومات من نطاق التطبيق الخاص بالنظام وليس من منطلق احتياج المستخدمين ، حيث تعكس هذه المعلومات خصائص وقيود هذا المجال مثل كيفية تنفيذ العمليات الحسابية **(Computations)** الخاصة بالنظام.

أما مخرجات عملية هندسة البرمجيات ، وكما هي موضحة بالشكل رقم 4.1 فهي كمايلي :

- **المتطلبات المُتفق عليها (Agreed Requirements)** : وهي عبارة عن وصف لمتطلبات النظام التي اتفق عليها ذوو الشأن أو أصحاب المصلحة **(Stakeholder)** من تطوير النظام.
- **مواصفات النظام (System Specification)** : وهي عبارة عن وصف تفصيلي للمتطلبات الوظيفية للنظام.
- **نماذج النظام (System Models)** : وهي مجموعة من النماذج التي تصف النظام مثل : نموذج سريان البيانات **(Data-Flow Model)** ،و نموذج العمليات **(Process Model)** ، وخلافه ، والتي تصف النظام من وجهات نظر مختلفة.

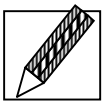
2. مفهوم المتطلبات

المتطلبات : تُعرف بأنها توصيف النظام الذي ينتج خلال عملية هندسة البرمجيات لخدمات النظام والقيود المفروضة عليه ، والتي يتفق عليها ذوو الشأن أو أصحاب المصلحة **(Stakeholder)** من تطوير النظام، **والمتطلبات** تتراوح بين كونها بيانات ملخصة عن الخدمات والقيودو بين كونها مواصفات تفصيلية تصف جميع الجوانب الهامة للنظام وبطريقة واضحة وموجزة ومترابطة وتأخذ بعين الاعتبار تحديد نوعية البيانات التي يجب معالجتها وكيفية إجراء هذه المعالجات وخصوصاً الحسابية منها **(Computation)** ، والوظيفة والأداء المطلوَبيْن والسلوك المتوقع من النظام ، والواجهات المطلوبة مع الأنظمة الأخرى ، والقيود المفروضة على التصميم ، ومؤشرات التحقق من صحة العمل المطلوبة لتعريف النظام الناجح.

وتُعد معرفة ما يؤديه النظام وكيفية أداء النظام لذلك ، وهو ما يُعرف "بالعلاقة ما بين التوصيف و التطبيق" ، أحد أكثر المسائل الجدلية أثناء عملية تحديد وتحليل المتطلبات، فيرى البعض أن توصيف المتطلبات يجب أن يتجنب مسائل التطبيق لعديد من الأسباب أهمها أنه من غير المتوقع أن يستوعب المستخدم المسائل الفنية للتطبيق ، وكذلك عدم وضع العقبات التي قد تعيق تطوير المشروع، ومن جانب آخر يرى البعض أنه من غير الممكن الفصل بين التوصيف والتطبيق وأنهما متداخلان ومترابطان في العديد من الطرق والأساليب الرئيسية. ففي تلك الطريقة ، يُعد استيعاب وتوثيق أعمال ومهام النظام المتاح فعلها (وقد يكون ذلك نظام يدوي أو نظام قائمة على الحاسب الآلي أو على كيهما) يعد أول مهمة في تطوير النظام الجديد، وبالتالي فإن تطبيق أحد الأنظمة يعد الجزء الأساسي في التوصيف لوضع نظام جديد. فمثلاً إذا افترضنا أنه لدينا الرغبة في تطوير ووضع

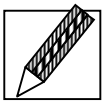
نظام آلي لأعمال الاختبارات ، فإحدى الطرق أن نقوم بالدراسة والتوثيق لكافة الإجراءات الحالية لأعمال الاختبارات الآلية منها واليدوية وعلى كافة المستويات ، وبعد الانتهاء من إنجاز هذه المهمة تأتي الخطوة الثانية كي نقرر الإجراءات يجب حوسبتها (أي إدخالها حاسوبياً) ، والتأكد من إمكانية تحقيق المطلوب من الناحية الفنية، وأخيرا فإن توصيف النظام الجديد مشتق من تصميم (تطبيق) النظام السابق أو القديم، وتبدو هذه الطريقة من التطوير منطقية وعملية جداً ، ولكن لا تعني أن التطبيق والتوصيف متداخلان.

تدريب (1)



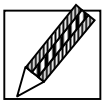
ما الفرق بين المتطلبات والتصميم ؟

تدريب (2)



ما الفرق بين متطلبات النظام ومتطلبات البرمجيات ؟

تدريب (3)



ما الفرق بين متطلبات العميل و متطلبات المطور؟

ولإلقاء مزيد من الضوء على مفهوم المتطلبات سوف نذكر بإيجاز بعضاً من المقارنات البسيطة التالية :

ما الفرق بين متطلبات العميل و متطلبات المطور؟

- متطلبات العميل : تُعد متطلبات إجبارية ، ويجب أن تُحقق في المنتج النهـــائي . وكمثال لمتطلبات العميل فإن البيانات يجب أن تخزن في ملف وبدون اســـتخدام أي قواعد بيانات خارجية.
- متطلبات المطور : تُعد متطلبات داعمة لتوقعات وطلبات العملاء ، ولنشـــاطات وخيارات التصميم ، وتنشأ هذه المتطلبات ــ عادة ــ من القيود المفروضة مـــن بيئة التطوير على النظام، وكمثال لمتطلبات المطور التي تدعم طلـــب العميـــل السابق فإن البيانات يجب أن تخزن في ملف فهرس متتالٍ لمناسبة هذا النوع من الملفات للنظام الذي يتم تطويره.

ولكن لماذا نحتاج المتطلبات ؟

- لتفهم أفضل للاحتياج وأسبابه لدى كل من المستخدمين والعملاء والمحلّلين.
- للحصول على المعلومات الضّروريّة والكافية لمراحل التّصميم والتّنفيذ اللاحقة.
- لاكتشاف المتطلبـــات المتناقضـــة **(Inconsistent Requirements)** والمشـــاكل المحتمل ظهورها أثناء مرحلة التنفيذ ، وذلك في مرحلة مبكرة.
- تقدير التّكلفة ، ومواعيد التنفيذ ، والموارد المطلوبة ، وخلافه.
- **لتفادي المشاكل التي قد تنشأ في مراحل التصميم والتنفيذ والصيانة ، والتـــي من أهمها ما يلي :**
  - ♦ المتطلبات لا تعكس الأهداف الرئيسية التي من أجلها تم تطوير النظام.
  - ♦ المتطلبات متناقضة أَو ناقصة.
  - ♦ زيادة التكلفة في حالة إحداث تغييرات للمتطلبات بعد الاتفاق عليها.
  - ♦ الفهم الخاطئ لمتطلبات المستخدمين والعميل من قبل فريق التطوير.
  - ♦ تجاوز الميزانية المقررة أو استغراق وقت أطول في عمليـــة التنفيـــذ مـــن التوقعات المبدئية لها.
  - ♦ معاناة فريق التطوير مِن إعادة العمل بسبب أخطاء أَو نقص في المتطلباتِ.

- ♦ افتقار الأنظمة الناتجة إلى بَعْض الوظائفِ أَو لَها وظائفُ غير مستعملةُ.
- ♦ ارتفاع تكاليف صيانة النظامِ.
- ♦ صعوبة تقويم تكلفة التغييرات أو تطوير النظام.

ولمن يتم تحديد المتطلبات ؟ وفيما تستخدمها ؟ :

يتم تحديد المتطلبات للاستفادة منها من قبل العديد من الأشخاص بمختلف خلفياتهم ومعرفتهم التقنية و على سبيل المثال العملاء ، والمستخدم النهائي ، و المحلّلون ، و المصمّمون ، و المهندسون ، و المديرون.

- **عملاء النظام ومستخدمون:** يستخدمون المتطلبات لفحصها للتحقق من مناسبتها لتلبية احتياجاتهم والمصادقة عليها و كذلك لإدخال التعديلات المطلوبة في حالة الضرورة.
- **محللي النظم :** وهم يستخدمون المتطلبات للتحقق من صحتها وتكاملها وتوافقها مع باقي مستندات المشروع.
- **مدراء المشروع :** وهم يستخدمون المتطلبات لغرض التخطيط للنظام و تطويره والتأكد من تسليم المشروع في الوقت المحدد وبمواصفات ومقاييسها الجودة المطلوبة.
- **مهندسي التصميم والتطوير :** وهم يستخدموا المتطلبات لتصميم وتطوير النظام بما يتلاءم معها ويحققها بالكامل.
- **مهندسو اختبار النظام :** وهم يستخدمون المتطلبات لتطوير مقاييس الاختبار التي سوف يستخدمونها في التحقق والمصادقة على صحة النظام.
- **مهندسو صيانة النظام :**وهم يستخدمون المتطلبات لمساعدتهم على فهم النظام والعلاقة بين أجزائه المختلفة لإجراء عمليات الصيانة اللازمة بكل سهولة وكفاءة وبأقل التكاليف.

ما هي الخصائص المميزة للمتطلبات الجيدة ؟

- **قابلة للإثبات (Verifiable)** : يجب أن تُعَرف المتطلبات بكلمات دقيقة في المعنى يسهل التحقق منها وقابلة للإثبات ، ولَيسَت مُعَرَّفَة بكلمات غير دقيقة يصعب التحقق من تنفيذها مثل : مقاومةِ **(Resistant)** ، كافيةِ **(Sufficient)** ، مفرطةِ **(Excessive)** ، .....الخ.
- **واضحة ومفهومه (Unambiguous and Understandable)** : أي أن جميع كلماتها وجملها تحتمل معنى واحداً فقط.
- **كاملة (Complete)** : تغطي جميع المهام المطلوبة من حيث : مبادئ الصيانة والتشغيل ، بيئات التشغيل ، والقيود المفروضة على كل مهمة ، وخلافه ، بحيث يمكن تنفيذها بدون حاجة المصممين والمطورين للاستفسار عن أي شيء فيها.
- **صحيحة (Correct)** : تؤدي في النهاية إلى تحقيق كامل متطلبات المستخدمين والعميل بصورة صحيحة.
- **محددة ومختصرة (Definitive and Abstractive)** : تحدد ما هو المطلوب ولماذا ، وذلك بكل دقة ووضوح وباختصار من خلال الإجابة الدقيقة على الأسئلة التي تبدأ بكلمة "ما هو/هي **(What)**" ، وكذلك الأسئلة التي تبدأ بكلمة "لماذا **(Why)**" ، وليس الإجابة على الأسئلة التي تبدأ بكلمة "كيف **(How)**".
- **متوافقة (Consistent)** : متّسقة مع المتطلباتِ الواردة في مستندات أخرى لنفس المشروع.
- **قابلة للتعديل (Modifiable)** : مكتوبة بطريقة منظمة بحيث يمكن تعديلها أو تغييرها بسهولة ويسر.

- **مجزأة (Partitioned) :** مقسمة إلى أجزاء مترابطة تتناسب مع تـــدرج مســـتوى هيكليـــة النظـــام **(System Hierarchy)** ، يمكـــن تتبعهـــا **(Traceable)** وفهمها بسهولة ويسر.
- **قابلة للاستخدام (Usable) :** قابلة للتنفيذ من قبل المصممين بسهولة ووضوح.

ما هي الفوائد الناتجة من تحديد المتطلبات بصورة جيدة؟

- خفض أو إزالة إعادة أنشطة التصميم وتسليم المنتج البرمجي.
- تحسين الجودة.
- التحكم في المشروع في جميع مراحله.
- تخفيض النزاعات مع العميل طوال وقت تنفيذ النظام.
- تخفيض التكاليف.
- الإيفاء بالجداول الزمنية لتنفيذ المشروع.
- إزالة احتمال حدوث فهم غير دقيق للمتطلبات.
- زيادة التواصل والتفاهم والتعاون والمساندة بين العميل وفريق التطوير.
- إزالة الاعتماد على أناس محددين في عملية تطوير المنتج لتخفيض احتمال المخاطرة عند تركهم لأعمالهم.

أسئلة تقويم ذاتي

ما هي الخصائص المميزة للمتطلبات الجيدة؟.

ما هي أهم الفوائد المترتبة على تحديد المتطلبات بالصورة الجيدة؟

3. أنواع المتطلبات (Types of Requirements)

بصفة عامة يمكن تقسيم المتطلبات إلى نوعين رئيسين هما :

- متطلبات وظيفية **(Functional Requirements)** .
- متطلبات غير وظيفية **(Non-Functional Requirements)** .

1.3 المتطلبات الوظيفية (Functional Requirements)

هي عبارة عن وصف للخدمات التي يجب أن تقدمها البرمجيـــة بالتفصـــيل ، وكيفية استجابتها لمدخلات محددة ، وكيفية سلوكها في حالات محددة ، وتعتمد هذه المتطلبات ــ في المقام الأول ــ على نوع البرمجية نفســـها وعلـــى المســـتخدمين النهائيين لها ، وكذلك على نوع النظام الداخلة في تكوينه والمتطلبـــات الوظيفيـــة – في بعض الحالات – ربما تتضمن ماذا يجب آلا يفعله النظام ، مثـــل : المتطلبـــات الخاصة بالأمان **(Security Requirements)**.

وتُعد المتطلبات الوظيفية هي الجوهر الأساسي لمواصفات المتطلبـــات أنهـــا تبين ماذا يجب أن يفعله النظام حيث إنها تتميز بالأفعال التي تـــؤدي عمـــلاً مـــا ، والأمثلة هي :

- النظام يجب أن يُظهر عناوين الكتب التي كتبها مؤلف معين.
- النظام يجب أن يُظهر باستمرار درجة حرارة كل المكينات.
- يجب أن يكون المستخدم قادراً على البحث في جميع قاعدة البيانـــات الفعليـــة الأولية أو اختيار مجموعة فرعية منها.
- كل مُدخلة يجب أن يخصص لها معرف مفرد **(ORDER_ID)** والتي تمكـــن المستخدم من التعامل معه بسهولة من حيث البحث والتخزين وخلافه.

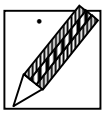
2.3 المتطلبات غير الوظيفية

. (Non-Functional Requirements)

هي عبارة عن وصف للقيود المفروضة على الخدمات التي يجب أن يقـــدمها النظام ، مثل التكلفة ،و تاريخ التسليم ،و كمية الذاكرة المتوفرة ، وكمية التخـــزين

المدعومة المتوفر ،و وقت الاستجابة، ولغة البرمجة وطريقة التطـوير المسـتخدمة **(Particular CASE Development System)** ،و أحجـام البيانـات ،و اعتماديـة النظام ،و قدرات وحدات الإدخال والإخراج، وخلافه. وقد تكون المتطلبـات غيـر الوظيفية أكثر أهمية من المتطلبات الوظيفية لأنه إذا لم يتم تحقيق المتطلبات غيـر الوظيفية فلا فائدة للنظام، وعادة تهتم المتطلبات غير الوظيفية بالخصائص النوعيـة **(Qualitative Features)** الخاصة بالمنتج البرمجي والتـي يمكـن قياسـها مثـل الاعتمادية **(Reliability)** ،و قابلية التنقل **(Portability)** ،و المتانة **(Robustness)** (قدرته على متابعة العمل في جميع الظروف حتى تلك غير المتوقعـة) ، والأمـان **(Security)** ،و سـهولة الاسـتخدام **(Ease of Use)** ،و سـهولة الصـيانة **(Maintainability)** ،و السرعة **(Speed)** ،و الحجم **(Size)**، وخلافه. والجدول رقم 4.1 يبين أهم تلك الخصائص وطرق قياسها.

تدريب (4)



في الحقيقة ، ليس من السهل التمييـز بـين المتطلبـات الوظيفية وغير الوظيفية ، بمعنى أنه يمكن أن تظهر متطلبـات وظيفية وكأنها متطلبات غير وظيفية ، اشرح هذه العبـارة مـع إعطاء مثال على ذلك

جدول رقم 4.1 : أهم خصائص وطرق قياس المتطلبات غير الوظيفية

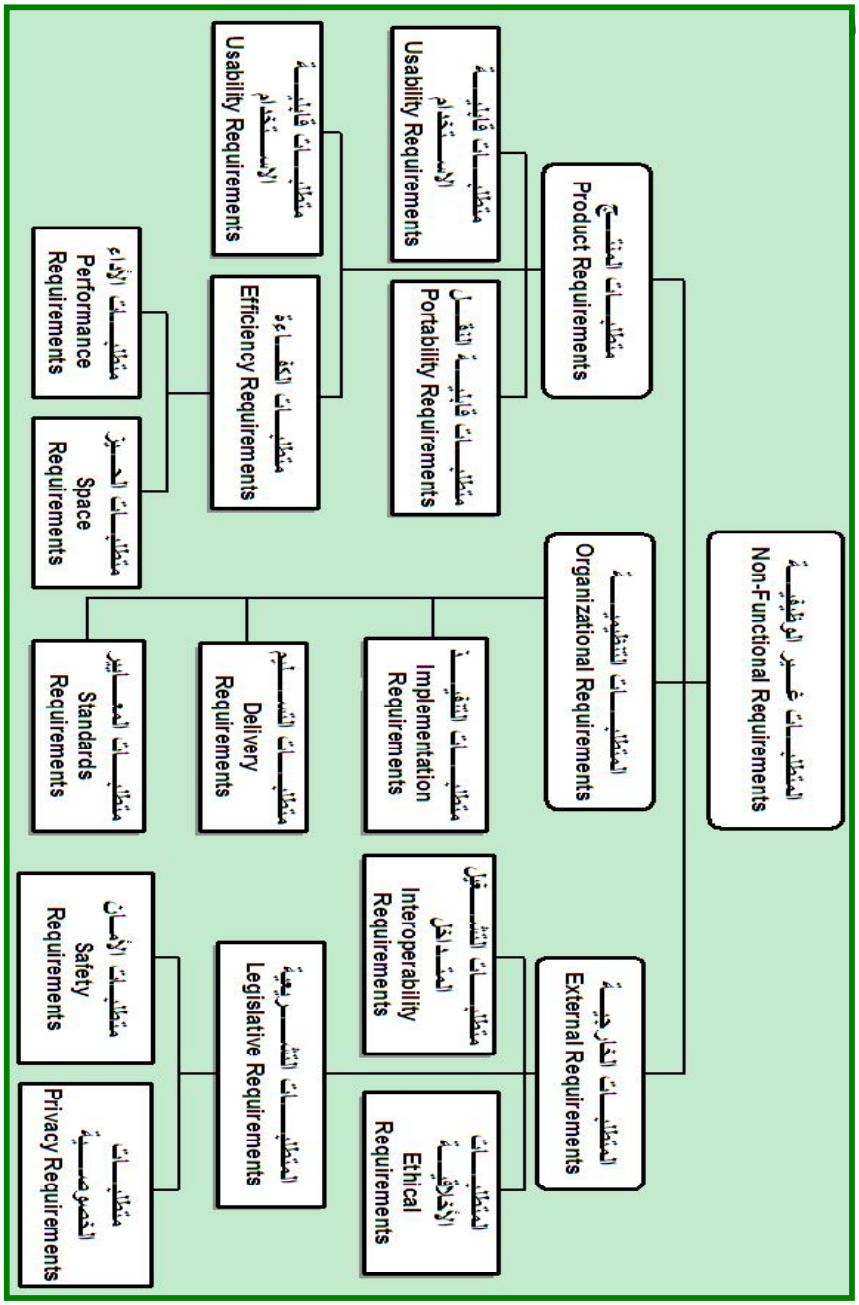
الخاصية	القياس
الاعتمادية	تقاس بـ : متوسط الوقت الذي يحدث من بعده عطـل أو فشـل فـي النظام ،و احتمالية عدم جاهزية النظام ،و معدل حدوث عطل أو فشـل

الخاصية	القياس
	في النظام ،و جاهزية النظام للعمل.
المتانة	تقاس بـــ : الوقت اللازم لتشغيل النظام بعد حدوث عطـــل أو فشـــل ، النسبة المئوية للخسارة نتيجة فشل النظام ،و احتمالية تخريب البيانـــات (Data Corruption) بسبب فشل النظام.
الحجم	ويقاس بسعة رقاقة الذاكرة العشوائية المطلوبة لتشغيل النظـــام (كيلـــو بايت). وكذلك يمكن أن يقاس بعدد سطور التعليمات البرمجيـــة التـــي يتكون منها النظام.
السرعة	تقاس بـــ : عدد الإجراءات التي تعالج في الثانية ، وقـــت الاســـتجابة للمستخدم أو لحدث ما ، وقت تجديد الشاشة.
سهولة الاستخدام	تقاس بـــ : وقت التدريب اللازم لإتقان العمل علـــى النظـــام ،وعـــدد أشكال المساعدة الفورية أثناء العمل على النظام.
قابلية النقل	تقاس بعدد أنظمة التشغيل التي يمكن تشغيل النظام من خلالها بدون أي تعديلات تذكر في التعليمات البرمجية الخاصة بالنظام.

ويمكن تصنيف المتطلبات غير الوظيفية ، كما هو موضح بالشكل 4.2 ، على النحو التالي :

- **متطلبات المنتج (Product Requirements) :** وهي المتطلبات التـــي تحـــدد سلوك المنتج البرمجي النهائي مثل : ســـرعة التنفيـــذ **(Execution Speed)** ، الاعتمادية(**Reliability**)الخ.
- **المتطلبات التنظيمية (Organizational Requirements) :** وهي المتطلبـــات الناتجة عـــن السياســـات والإجـــراءات التنظيميـــة **(Organizational Policies and Procedures)** الخاصة بالشركة أو المؤسسة التي يطورها لها المنتج البرمجـــي ، مثـــل :

المعايير القياسية المستخدمة في عمليـــة التطـــوير **(Process Standards)** ، ومتطلبـــات



شكل 4.2 : شكل توضيحي لأهم أنواع المتطلبات غير الوظيفية.

- **المتطلبات الخارجية : (External Requirements)** : وهي المتطلبات التـــي تنشأ من عوامل خارجية للنظام ولعملية التطوير الخاص بها ، مثل المتطلبات

التشريعية فمثلاً يجب أن لا يفشي النظام أية معلومات شخصية عن العمـــلاء بغض النظر عن أسمائهم وأرقام هواتفهم) إلى مشغلي النظام.

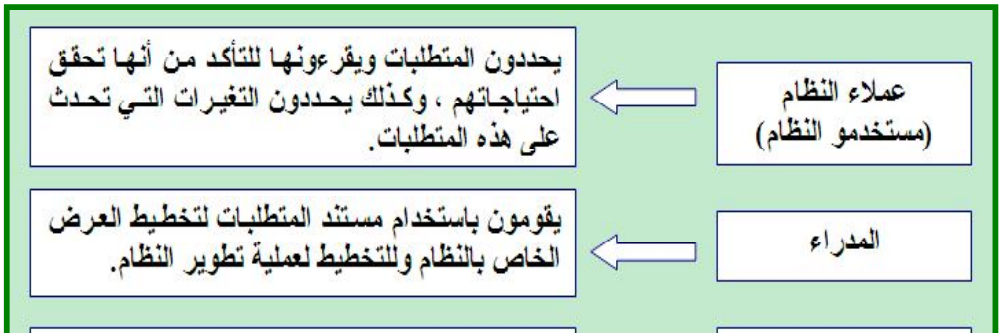
وقد تكون المتطلبات غير الوظيفية صعبة جداً بحيث لا يمكن وصفها بدقـــة ، ولكن المتطلبات غير الدقيقة (الغامضة) من الصعب التأكد من صحتها ، فعلى سبيل المثال عند تحديد الهدف من النظام بـــالقول " **يجـــب أن يكـــون النظـــام ســـهل الاستعمال من قبل مراقبين ذوي خبرات ، وكذلك يجب أن يكون منظماً بحيث يقلل من أخطاء المستخدمين**" بالتدقيق في منطوق هذا الهدف نجد أنه غـــامض وغيـــر قابل للقياس والتحقق من تنفيذه ، وحيث إن تحديد أهداف النظام مفيـــدة للمطـــورين لأنها تظهر الرغبات والاهتمامات الحقيقية لمستخدمي النظام ، لذا يجب أن يكـــون منطوق الهدف واضحاً وقابلاً للقياس ، فمثلاً يمكن تعديل المنطـــوق الســـابق إلـــي منطوق واضح وقابل للقياس بالقول : " **يجب أن يكون المراقبين من ذوي الخبـــرة قادرين على استخدام جميع وظائف النظام بعد تدريب حوالي ساعتين وبعـــد هـــذا التدريب فإن متوسط عدد الأخطاء التي يرتكبها المستخدمين من ذوي الخبرة يجب أن لا تزيد عن خطأين في اليوم** ".

4. وثيقة مواصفات المتطلبات

(Requirements Specification Document)

المنتج النهائي لعملية هندسة المتطلبات هو وثيقة مواصفات المتطلبات ، وهي ضرورية لنجاح وتطوير أي مشروع تطوير برمجية، فإذا لم نحدد بوضوح ما يستطيع أن يفعله النظام فكيف يمكننا أن نطور البرمجية وبأي ثقة. وكيف نأمل أن يتوافق المنتج النهائي مع احتياجاتنا؟ . ووثيقة المواصفات هي الوثيقة التي يمكن الرجوع إليها عند تقويم أي تطوير ، وهي عنصر رئيسي لأي عقد ارتباط شرعي، ومن أهم العوامل التي يجب مراعاتها عند كتابة وثيقة مواصفات المتطلبات هو الأخذ بعين الاعتبار مستخدم هذه الوثيقة (أنظر شكل 4.3) ، لكي يفهمها كل من له صلة في تطوير النظام باختلاف مستوياتهم من حيث الخلفيات العلمية والخبرات العملية ، وخصوصاً المستخدمين والمطورين.

وبرغم أن كلاً من هؤلاء يشتركون في الهدف المشترك وهو الوصف الواضح لمتطلبات النظام ، فإنهم سيضطرون لاستعمال لغات مختلفة ، حيث إن خلفياتهم العلمية وخبراتهم العملية مختلفة، فمثلاً سيفضل المستخدمون وصفات غير تقنية بلغة طبيعية ، وبالرغم من أن تلك اللغات تكون ممتازة في القصائد وكتابة الخطابات ولكنها فقيرة وضعيفة في عرض المواصفات الدقيقة والمتكاملة والواضحة ، من الناحية الأخرى فإن المحللين لكونهم أصحاب خبرة تقنية سيفضلون استخدام رموز دقيقة (ربما رياضية) لكي يصفوا النظام.



هناك رموز متعددة لكتابة المواصفات:

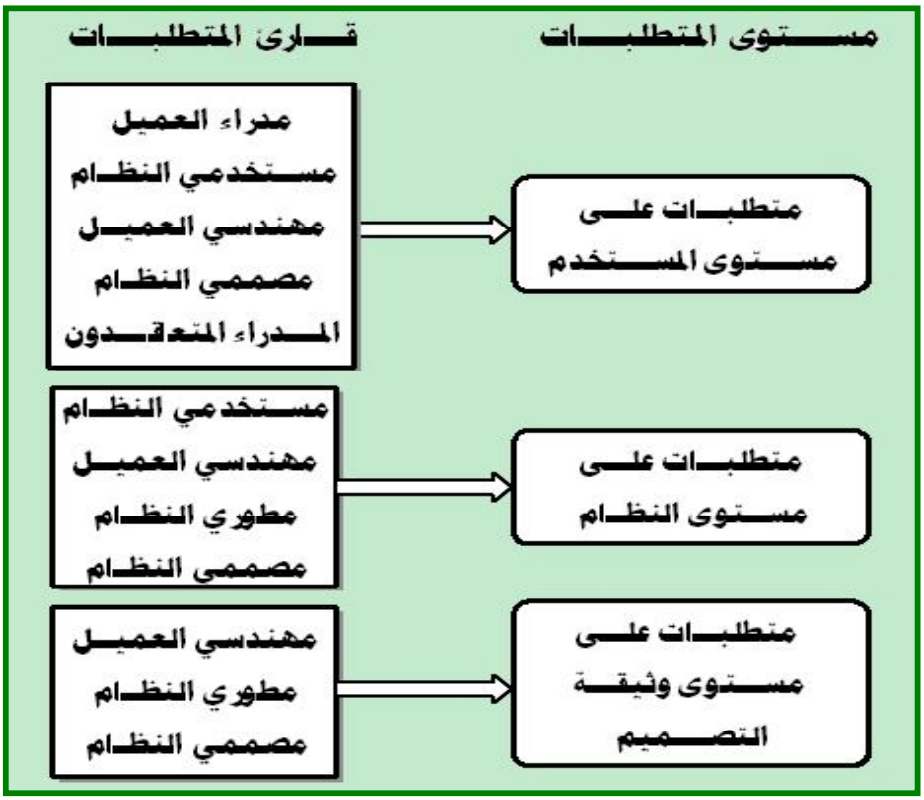
- الكتابة غير الرسمية **(Informal)** : وهي تتم باللغة الطبيعية حيث ستستخدم بوضوح وبعناية قدر الإمكان.
- الكتابة الرسمية **(Formal)** : حيث تستعمل الرموز الرياضية بكل دقة وباختصار.
- الكتابة شبة رسمية **(Semiformal)** : حيث تتم باستعمال مزيج من اللغة الطبيعية مع رموز وجداول ورسوم مختلفة.

ومعظم هذه الرموز لديها أصول في الطرق لتصميم برمجية حيث تشمل الشفرة الكاذبة **(Pseudo Code)** ، ورسوم تدفق البيانات **(Flow Charts)** ، وخلافه. وفي الوقت الحالي فإن معظم مواصفات البيانات تكتب بلغة طبيعية مع

رموز شبه رسمية مثل رسومات تدفق البيانات المشروحة حيث تستخدم أحياناً لتوضيح نص اللغة الطبيعية.

1.4 مستويات المتطلبات (Levels of Requirements)

لتوصيف متطلبات النظام بصورة جيدة يفهمها كل من يستخدم هذه الوثيقة بمختلف مستوياتهم الوظيفية وخبراتهم العملية يجب تقسيم وثيقة مواصفات المتطلبات إلى مستويات مختلفة من التفاصيل لكي تتناسب مع الاهتمامات الخاصة لكل من يقرأها ، مثل المستخدم النهائي و المدير التنفيذي و المطور ، وخلافه ، كما هو موضح بالشكل رقم 4.4.



شكل 4.4 : شكل توضيحي لأهم مستويات المتطلبات وقارئها.

وم

- متطلبات المستخدم **(User Requirements)** .

متطلبات المستخدم للنظام ينبغي أن تصف المتطلبات الوظيفية وغير الوظيفية لكي تكون مفهومة لمستخدمي النظام بدون المعرفة التقنية المفصلة له، ويجب أن تشمل سلوك النظام الخارجيّ وتجنب بقدر الإمكان خواص تصميم النظام، ويراعى عند كتابة متطلبات المستخدم أن لا تستعمل المصطلحات البرمجية (نظم الترميز) الرّسميّة أو وصف المتطلبات بواسطة وصف تنفيذ النظامو ينبغي أن تكتب متطلبات المستخدم باللغة البسيطة ، وبجداول وأشكال بسيطة ورسوم بيانية واضحة.

وقد تظهر العديد من المشاكل المختلفة عندما تكتب المتطلبات بجمل لغة طبيعية في نصّ المستند للأسباب التالية :

① **نقص الوضوح (Lack of Clarity) :** أحيانا يكون من الصعب استخدام اللغة الطبيعية بطريقة دقيقة وواضحة بدون جعل الوثيقة مفصلة وصعبة القراءة.

②**التشويش بين المتطلبات (Requirements Confusion) :** حيث أنه يكون من الصعب التميز بوضوح بين المتطلبات الوظيفيّة وغير الوظيفيّة ، وكذلك يبين أهداف النظام ومعلومات التصميم.

③ دمج المتطلبات **(Requirements Amalgamation) :** حيث إنه يتم التعبير عن عدّة متطلبات مختلفة كمتطلب واحد.

ولتقليل سوء الفهم عند كتابة متطلبات المستخدم ، يوصي البعض باتباع التوجيهات البسيطة التالية :

① تحديد شكل قياسيّ لكتابة المتطلبات والتأكد من أنّ كلّ تعريف للمتطلب ينتسب إلى ذلك الشكل و توحيد الشّكل يجعل المحذوفات أقلّ احتمال والمتطلبات أسهل للفحص، فعلى سبيل المثال يعرض الشكل المتطلب الأولي بخطّ كبير، والتفاصيل بخطوط مختلفة متضمّنًة أسباب كل متطلب ومعلومات عن مقترح المتطلب (مصدر المتطلب) ، لكي تعرف من تستشير في حال تغير المتطلبات.

② استعمل لغة على نفس النّهج ، وينبغي أن تميّز دائمًا بين المتطلبات الإجبارية والمفضّلةو المتطلبات الإجباريّة هي المتطلبات التي يجب على النّظام أن يُدعمها ، أما المتطلبات المفضلة هي المتطلبات المرغوب فيها ولكنها غير أساسيّة.

③استخدم إبراز النصّ (أسود عريض ، مائل أو لون) لتلتقط الأجزاء الأساسيّة من المتطلبات.

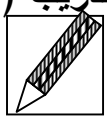
④تجنب بقدر الإمكان ، استعمال المصطلحات العلمية التقنية لعلوم الحاسب الآلي ، وفي حالة استخدامها يجب تعريفها بدقة ووضوح.

- متطلبات النظام **(System Requirements)** .

تُعد متطلبات النظام امتداد لمتطلبات المستخدم والتي يستخدمها مهندسوا البرمجيات كنقطة انطلاق لتصميم النظام حيث تضيف تفاصيلاً وشرحاً لكيفية تناول متطلبات المستخدم ، ويمكن استخدامها كجزء من العقد لاستخدام النظام ، لذا يجب أن تكون كاملة ومتوافقة مع النظام الكلي.

وعموما فإن متطلبات النظام تصف ببساطة سلوك النظام الخارجي والقيود الموضوعة عليه ولا يجب أن تتطرق إطلاقا إلى كيف يتم تصميم النظام. وتستخدم اللغة الطبيعية لكتابة كل من مواصفات متطلبات النظام وكذلك متطلبات المستخدم حيث تُعد مواصفات النظام أكثر تفصيلا عن متطلبات المستخدم.

تدريب (5)



اشرح الصعوبات في استعمال اللغة الطبيعية لوصف المتطلبات.

وعموما فإنه يحبذ كتابة متطلبات المستخدم بلغة يفهمها غير المتخصصين في حين يمكن كتابة متطلبات النظام ، بحيث تشتمل على أنماط مختلفة كالمبينة

بالجدول رقم 4.2 ، مثل استخدام : قوالب للغة الطبيعية ، ولغات وصف برمجية ، والبيانات الرسومية ، والتمثيل الرياضي.

جدول رقم 4.2 : بعض الأنماط لوصف المتطلبات

النمط	الوصف
استخدام القوالب للغة الطبيعية	**يعتمد على تعريف ثوابت وقالب للتعبير على مواصفات المتطلبات.**
استخدام لغات الوصف	**يتم فيه استخدام لغة برمجة معينة ولكن بطريقة موجزة للتعبير عن المتطلبات بوضع موديل تطبيقي للنظام.**
استخدام الرسومات	**تستخدم اللغة الرسومية المزودة بالنصوص لتعريف المتطلبات الوظيفية للنظام.**
الوصف الرياضي	**هناك بعض الملحوظات التي تعتمد على الوصف الرياضي حيث تزيل إلى حد كبير أي خلافات بين العميل والمنفذ للنظام وبالرغم من أن هذه المواصفات تكون صعبة الفهم بالنسبة للعميل ولا تهم العميل قدر ما تهم منفذ النظام.**

- مواصفات تصميم البرنامج **(Software Design Specification)** :

وهي مواصفات مفصلة للبرنامج والتي قد تستخدم كقاعدة للتصميم أو التطبيق ، وتكتب للمطورين.

2.4 بنية مستند مواصفات المتطلبات

(The Structure of Requirements Specification Document) :

يوجد العديد من المعايير القياسية لكتابة مستند مواصفات المتطلبات ، والتي

من أهمها : المعيار المقدم من المنظمة العالمية **IEEE** والمعروف باسم **IEEE/ANSI 830-1998:** ، والشكل 4.5 يوضح أهم المحتويات الأساسية لمستند مواصفات المتطلبات بناءً على معيار المنظمة العالمية **IEEE** .

الفصل	الوصف
التمهيد (Preface)	يجب أن تحدد مجموعة القراء المتوقعين للوثيقة ، وكذلك تقديم وصف تاريخي للإصدارات السابقة لها، بما في ذلك عرض للمبادئ الأساسية لتطوير هذه النسخة الجديدة ، وخلاصة للتغييرات والإضافات التي تم إجرائها في كل نسخة بما في ذلك النسخة الجديدة.
المقدمة (Introduction)	يجب أن تصف دواعي الحاجة للنظام ، وتعطي فكرة موجزة عن وظائفه وتبين كيف يعمل مع الأنظمة الأخرى. ويجب أن توضح كيف يتوافق النظام مع الأعمال عموماً أو الأهداف الإستراتيجية للجهة المفوضة بتطوير البرمجية.
مسرد المصطلحات (Glossary)	يجب أن تعرّف المصطلحات التقنية المستخدمة في الوثيقة ، لأنه من الخطأ أن تفترض معرفة القارئ بهذه المصطلحات.
تعريف متطلبات المستخدم (User Requirements Definition)	في هذا الفصل يجب أن تصف الخدمات المقدمة للمستخدم والمتطلبات الوظيفية وغير الوظيفية للنظام على مستوى المستخدم ، و حتى يكون أكثر استيعاباً فإنه يجب عليك استخدام لغة مألوفة للعملاء ورسوم توضيحية وغيرها من الرموز المفهومة، وكذلك يجب إتباع المعايير القياسية للمنتج والعمليات التي ينبغي إتباعها.
بنية النظام (System Architecture)	يجب أن يقدم في هذا الفصل فكرة عامة وافية عن بنية النظام المُستعمل، مبيناً توزيع الوظائف عبر نماذج النظام ، و كذلك يجب إبراز عناصر البنية التي يعاد استخدامها.
توصيف متطلبات النظام (System Requirements Specification)	في هذا الفصل يجب أن تقدم وصف مفصل عن المتطلبات الوظيفية وغير الوظيفية ، وإذا لزم الأمر ، يمكن أن تضيف تفاصيل أخرى عن المتطلبات غير الوظيفية مثل: تعريف واجهات الأنظمة.
نماذج النظام (System Models)	في هذا الفصل يجب أن توضح نموذج أو أكثر من نموذج للنظام مبيناً العلاقات بين مكونات النظام والنظام وبيئته ، والتي قد تكون نماذج الكائن ، ونماذج تدفق البيانات، ونماذج البيانات الدلالية.
تطور النظام (System Evolution)	في هذا الفصل سيتم توضيح الافتراضات الأساسية التي قام عليها النظام والتغييرات المتوقعة بسبب التطور في العتاد ، والتغيرات في احتياجات المستخدم.... الخ..
الملاحق (Appendices)	في هذا الفصل يجب أن تقدم المعلومات التفصيلية المتعلقة بالتطبيق الذي يطور، ومن أمثلة الملاحق التي يمكن إضافتها أوصاف العتاد وقواعد البيانات ، و بالخصوص متطلبات العتاد الدنيا والمثلى لتشغيل النظام ، وكذلك وصف متطلبات قاعدة الخاصة بالتنظيم المنطقي للبيانات المستخدمة من قبل النظام والعلاقات بينها.
الفهرس (Index)	يمكن أن تضاف عدة فهارس للوثيقة مثل : فهرس الرسوم التوضيحية والمخططات وفهرس الوظائف وغيرها من الفهارس. بالإضافة إلى الفهرس الأبجدي.

شكل 4.5 : بنية مستند مواصفات المتطلبات المبني على المعيار المقدم من IEEE .

الخلاصة

اشتملت هذه الوحدة على:

– تعريف هندسة المتطلبات (عملية تتضمن مجموعة من الأنشطة الهيكلية التي يتم إتباعها لاستنتاج وثيقة متطلبات النظام والمصادقة عليها).

*مراحل (أنشطة) هندسة متطلبات البرمجيات:

– مرحلة استنباط المتطلبات وهي مسؤولية كل من المحللين والمستخدمين بالحوار مع بعضهم بحيث يحاول كل منهم أن يتفهم الأخر ، مع الإشارة إلى أهم الإرشادات التي يجب مراعاتها في هذه المرحلة.

المرحلة الثانية: التفاوض حول المتطلبات وتحليلها، وهنا يتم تصنيف المتطلبات وتنظيمها في مجموعات واكتشاف العلاقات بينها وبين المتطلبات الأخرى.

المرحلة الثالثة: توثيق المتطلبات، وهي كتابة بيان واضح غالباً بلغة طبيعية عما يتوقع أن يقدمه النظام للمستخدمين.

المرحلة الرابعة: المصادقة على المتطلبات تؤدي هذه المرحلة إلى إنتاج تقرير عن المصادقة عن المتطلبات ، وبناء على هذا التقرير أما أن تقبل مسودة الوثيقة النهائية للمتطلبات المنتجة في المرحلة السابقة وتصبح وثيقة نهائية أو يتم إعادة تكرار العملية مرة أخرى.

* مفهوم المتطلبات: تعرفنا هنا على بعض المقارنات:

الفرق بين المتطلبات والتصميم ، الفرق بين متطلبات النظام ومتطلبات البرمجيات ، الفرق بين متطلبات العميل ومتطلبات المطور.

وقد أجبنا على الأسئلة الآتية: لماذا نحتاج المتطلبات؟ لمن يتم تحديد المتطلبات؟ وفيما نستخدمها؟

* أنواع المتطلبات:

وظيفية: وهي عبارة عن وصف للخـــدمات التـــي يجـــب أن تقـــدمها البرمجيـــة بالتفصيل.

غير وظيفية: وهي عبارة عن وصف للقيود المفروضة على الخدمات التي يجب أن يقدمها النظام.

- وثيقة مواصفات المتطلبات: وقد عرفنا إنها ضـــرورية لنجـــاح وتطوير أي مشروع تطوير برمجية ، وتعرفنا علـــى مســـتويات المتطلبات ، بنية مستند مواصفات المتطلبات.

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس،،

بعد أن فرغت من دراسة هندسة متطلبات البرمجيات في الوحدة الرابعة ينتقل بك المقرر إلي الوحدة الخامسة التي سنتابع فيها التحقق والمصادقة على صحة البرمجيات ، وفيها يتم بيان الفرق بين التحقق من صحة البرمجيات وبين المصادقة عليها، وسنتعرف فيها على طبيعية الأخطاء الأكثر شيوعاً وانتشاراً في البرامج.

نرجو أن تكون وحدة مفيدة لك وأن تساهم معنا في نقدها.

إجابات التدريبات

تدريب (1)

- المتطلبات يجب أن تصف : ماذا يجب على النظام أن يفعل؟ .
- التصميم يجب أن يصف : كيف يتم بناء النظام ليحقق متطلبات العميل؟ .

تدريب (2)

- متطلبات النظام تشمل القيود والإمكانيات لكامل المنتج بما فيها كيان المكونات المادية **(Hardware)** والكيان البرمجي **(Software)**.
- متطلبات البرمجيات هي جزء من متطلبات النظام.
- متطلبات النظام غالبا ما تكون مقيدة بواسطة السياسات التنظيمية والبيئية للمؤسسة.
- متطلبات البرمجيات ربما تكون وربما لا تكون مقيدة من قبل السياسات التنظيمية للمؤسسة.

تدريب (3)

- متطلبات العميل : تُعد متطلبات إجبارية ، ويجب أن تُحقق في المنتج النهائي . وكمثال لمتطلبات العميل : البيانات يجب أن تخزن في ملف وبدون استخدام أي قواعد بيانات خارجية.

متطلبات المطور : تُعد متطلبات داعمة لتوقعات وطلبات العملاء ، ولنشاطات وخيارات التصميم ، وتنشأ هذه المتطلبات ــ عادة ــ من القيود المفروضة من بيئة التطوير على النظام، وكمثال لمتطلبات المطور التي تدعم طلب العميل السابق : البيانات يجب أن تخزن في ملف فهرس متتالٍ لمناسبة هذا النوع من الملفات للنظام الذي يتم تطويره

تدريب (4)

فعلاً إنه من الصعب التمييز بين بعض المتطلبات الوظيفية وغير الوظيفية من الوهلة الأولى وخصوصاً التي تخص أمن النظام ، ولكن الفرق يتضح شيئاً فشيئاً

أثناء عملية التطوير والتي يتم خلالها دراسة جميع تفاصيل هذه المتطلبات والمتطلبات الجديدة التي تنشأ لتنفيذها والتي في حد ذاتها متطلبات وظيفية ، فمثلاً متطلبات المستخدم المعنية بأمن النظام والتي تُعد من المتطلبات الوظيفية تبدو وكأنها متطلبات غير وظيفية مثل الحاجة إلى تضمين نظام التحقق من شخصية المستخدم قبل الدخول للنظام عن طريق اسم مستخدم وكلمة مرور (User Authentication System) .

تدريب (5)

تفاصيل اللغة الطبيعية يمكن أن تكون أكثر تشويشا ويصعب فهمها للأسباب التالية:

1. يعتمد فهم اللغة الطبيعية على القراء والذين يكتبون المواصفات إذا كانوا يستعملون نفس الكلمات بنفس المفهوم.

2. مواصفات اللغة الطبيعية تعتبر مطاطة وأكثر مرونة حيث يمكن قول نفس الشيء بطرق مختلفة.

3. ليس هناك طريقة لوضع قوالب ثابتة محددة للغة الطبيعية.

مسرد المصطلحات

هندسة المتطلبات :

تُعرف هندسة المتطلبات بأنها عملية (Process) تتضمن مجموعة من الأنشطة الهيكلية (Structured Set of Activities) التي يتم إتباعها لاستنتاج وثيقة متطلبات النظام والمصادقة عليها ، وتوثيقها ، والمحافظة عليها.

أسلوب المقابلات Interviews Technique :

عملية اتصال مباشر ما بين المستخدمين ومن لهم صله مباشرة بالنظام (Stakeholders) ومصممي أو مهندسي البرمجيات.

قوائم تدقيق المتطلبات Requirements Checklists:

وتستخدم للتأكد من عدم نسيان أي من المتطلبات في جميع مراحل هندسة المتطلبات ، وكذلك التأكد من وضوح المتطلبات وقابليتها للاختبار وعدم وجود أية تعارضات أو تكرارات أو مرادفات بين المتطلبات.

أصحاب المصلحة Stakeholder:

أفراد أو مجموعات معينة لها تأثير هام على أو يمكن أن تتأثر أو لها مصلحة مباشرة بـ "مجموعة من الأنشطة" المتعلقة بتطوير منتج ما ، أي أنهم هم الأشخاص المهتمين بإنجاز عملية تطوير المنتج.

التعليمات التنظيمية Regulations :

هي التعليمات والقيود المفروضة على النظام مثل تعليمات الصحة ، والأمن التي تطبق على النظام.

المتطلبات المُتفق عليها Agreed Requirements :

وهي عبارة عن وصف لمتطلبات النظام التي اتفق عليها ذوي الشأن أو أصحاب المصلحة (Stakeholder) من تطوير النظام.

مواصفات النظام System Specification :

وهي عبارة عن وصف تفصيلي للمتطلبات الوظيفية للنظام.

نماذج النظام System Models :

مجموعة من النماذج التي تصف النظام مثل : نموذج سريان البيانات (Data-Flow Model) ، نموذج العمليات (Process Model) ، وخلافه ، والتي تصف النظام من وجهات نظر مختلفة.

مفهوم المتطلبات :

المتطلبات تُعرف بأنها توصيف النظام الذي ينتج خلال عملية هندسة البرمجيات لخدمات النظام والقيود المفروضة عليه ، والتي يتفق عليها ذوي الشأن أو أصحاب المصلحة (Stakeholder) من تطوير النظام.

المتطلبات الوظيفية Functional Requirements :

هي عبارة عن وصف للخدمات التي يجب أن تقدمها البرمجية بالتفصيل ، وكيفية استجابتها لمدخلات محددة ، وكيفية سلوكها في حالات محددة ، وتعتمد هذه المتطلبات ــ في المقام الأول ــ على نوع البرمجية نفسها وعلى المستخدمين النهائيين لها ، وكذلك على نوع النظام الداخلة في تكوينه.

المتطلبات غير الوظيفية Non-Functional Requirements :

هي عبارة عن وصف للقيود المفروضة على الخدمات التي يجب أن يقدمها النظام ، مثل : التكلفة ، تاريخ التسليم ، كمية الذاكرة المتوفرة ، كمية التخزين المدعومة المتوفر ، وقت الاستجابة ، لغة البرمجة وطريقة التطوير المستخدمة (Particular

(CASE Development System ، أحجام البيانات ، إعتمادية النظام ، قـدرات وحدات الإدخال والإخراج ، ....وخلافه.

المتطلبات الخارجية External Requirements :

وهي المتطلبات التي تنشأ من عوامل خارجية للنظام ولعملية التطوير الخاص بهـا ، مثل : المتطلبات التشريعية (فمثلاً : يجب أن لا يفشـي النظـام أيـة معلومـات شخصية عن العملاء (بغض النظر عن أسماءهم وأرقام هـواتفهم) إلـى مشـغلي النظام.

المصطلح بالإنجليزية	معناه بالعربية
Agreed Requirements	المتطلبات المُتفق عليها
Analyst as Apprentice	أسلوب المحلل كمتدرب
Analytic Hierarchy Process	عملية التدرج التحليلي
Customer Requirements	متطلبات العميل
Data Mining Technique	أسلوب استخلاص البيانات
Developer Requirements	متطلبات المطور
Domain Information	معلومات المجال
External Requirements	المتطلبات الخارجية
Formal Modeling	النمذجة الاصطلاحية (المنهجية)
Functional Requirements	المتطلبات الوظيفية
Informal Modeling	النمذجة غير الاصطلاحية
Interviews Technique	أسلوب المقابلات
Levels of Requirements	مستويات المتطلبات
Non-Functional Requirements	المتطلبات غير الوظيفية
Observation and Social Analysis	أسلوب التحليل الاجتماعي والمشاهدة
Organizational Requirements	المتطلبات التنظيمية
Organizational Standards	المعايير التنظيمية
Product Requirements	متطلبات المنتج
Regulations	التعليمات التنظيمية
Requirements Change Management	إدارة تغيير المتطلبات
Requirements Checklists	قوائم تدقيق المتطلبات
Requirements Elicitation	استنباط المتطلبات
Requirements Prioritization	تعيين أولوية المتطلبات

المصطلح بالإنجليزية	معناه بالعربية
Requirements Reuse	إعادة استخدام المتطلبات
Requirements Reviews	فحص المتطلبات
Requirements Specification Document	وثيقة مواصفات المتطلبات
Requirements Traceability	تتابعية المتطلبات
Requirements Validation Report	المصادقة على المتطلبات
Semi-Formal Modeling	النمذجة شبه الاصطلاحية
Software Design Specification	مواصفات تصميم البرنامج
Software Requirements Engineering	هندسة متطلبات البرمجيات
Stakeholder	ذوو الشأن أو أصحاب المصلحة
Structure of Requirements Specification Document	بنية مستند مواصفات المتطلبات
System Models	نماذج النظام
System Requirements	متطلبات النظام
System Specification	مواصفات النظام
User Requirements	متطلبات المستخدم

المراجع

أولاً : المراجع العربية :

[1] مهندس عبد الحميد بسيوني ، "أساسيات هندسة البرمجيات" ، دار الكتب العلمية للنشر والتوزيع ، القاهرة ، 2005 م.

ثانياً : المراجع الإنجليزية :

[2] Ian Somerville , "Software Engineering", Addison Wesley, 2001.

[3] Ronald J. Leach,, "Introduction to Software Engineering", CRC Press, 1999.

[4] Douglas Bell , "Software Engineering A Programming Approach", 3rd Edition, Addison Wesley.

[5] **Shari Pfleeger, "Software Engineering - Theory and Practice", 2nd Edition.**

ثالثاً : مواقع على شبكة الإنترنت تم الاستفادة منها :

[6] www.cs.uwlax.edu/~zheng/CS341Spring05/Lecture3.ppt,
"What is a software requirement?".

[7] www.comp.lancs.ac.uk/computing/ resources/re/slides/Chapter6.ppt,
" Requirements Engineering Processes"

[8] www.elearn.cerc.wvu.edu/691K/slides/lecture%201.ppt,
"Software Requirements: A Beginning"

[9] www.soc.staffs.ac.uk/ch14/PPSE/ReqSpec1.ppt ,
"Requirements specification"

[10] www.nknucc.nknu.edu.tw/~dmyeh/se/chap11.ppt ,
" Software Requirement Engineering"

[11] www.academics.vmi.edu/.../EE321/EE321_course_
PowerPoint_presentations/Requirements_analysis_short_version.ppt ,
" Requirements Analysis "

[12] www.julianjewel.com/Julian/ articles/jan_1999/rm_aad_5.ppt ,
" Software Requirements "

[13] www.cems.uwe.ac.uk/~sjgreen /
UQC139RequirementsEngineering/Lectures/LectureTwo/Lecture_two.ppt ,
" What is RE?"

[14] www.comp.lancs.ac.uk/computing /resources/re/slides/Ch2REproc.ppt
" Requirements Engineering Processes"


الوحدة الخامسة
التحقق والمصادقة
على صحة البرمجيات

محتويات الوحدة

مقدمة

تمهيد

عزيزي الدارس،،

أهلاً بك في الوحدة الخامسة من مقرر "مقدمة في هندسة البرمجيات" وهي تتناول بيان الفرق بين التحقق من صحة البرمجية وبين المصادقة عليها ومعرفة طبيعة الأخطاء الأكثر شيوعاً وانتشاراً في البرامج.

في هذه الوحدة سنتناول:

* طبيعة الأخطاء وفيها نتناول أهم أنواع الأخطاء التي قد تظهر عند تشفير (تكويد) احد المكونات وهي عدم إعطاء قيم بدائية للبيانات و تكرار الحلقات عدد مرات غير صحيح و أخطاء في القيم الحدية.

* التحقق والمصادقة الساكنة والديناميكية وتشمل طريقتين: فحص البرنامج ، واختيار البرنامج.

* اكتشاف وتصحيح الأخطاء وهي العملية التي تحدد مكان الأخطاء حيث يتم تصحيحها.

* التخطيط للتحقق والمصادقة وهنا سنتعرف على نموذج الاختيار والتطوير وسنوضح فيه كيفية استخراج خطط الاختيار من مواصفات النظام وفق تصاميمه.

* فحوصات البرامج وهي مراجعات هدفهااكتشاف الأخطاء التي قد تكون موجودة في البرنامج ، وسنتعرف هنا أيضاً على فرق التفتيش والشروط المسبقة للتفتيش ،وعملية التفتيش ، وقوائم الفحص ، ومعدلات التفتيش.

* التحليل الآلي الساكن:هو أدوات برمجية تمر على جميع محتويات نص البرنامج المصور وتكشف الأخطاء التي فيه ، وسوف نتناول في هذا القسم أيضاً فحوصات التحليل الساكن ،ومراحل التحليل الساكن ،و استخدامات التحليل الساكن.

* اختبار الصندوق الأسود: ابتكار عينة من البيانات تنوب عن (تمثل) كل البيانات المحتملة.

قواعد اختيار بيانات الصندوق الأسود باستخدام تكافؤ التقسيم هي:

- اختبار بيانات تمثل الكل من كل جزء أو قسم.
- اختبار بيانات متاخمة (بالقرب من) للأقسام.

_ اختبار الصندوق الأبيض: يستغل اختبار الصندوق الأبيض معرفة كيف يعمل الإجراء (procedure) حيث يستخدم قوائم شفرة البرنامج.

- اختبار بيتا: يتم طرح احد الإصدارات التمهيدية من البرنامج للعميل والذي يكون على دراية بأن المنتج به عيوب.
- اختبار اندماج النظام: توجد ثلاثة اتجاهات لاختبار النظام هي:

* الاختبار الكبير.
* الاختبار الكبير المطور.
* الاختبار المتزايد.

غير أن هناك طريقتين للاختبار المتزايد هما: الاختبار من الأسفل إلى الأعلى (التصاعدي)، والاختبار من القمة إلى الأسفل (التنازلي).

- تطوير البرمجيات بطريقة الغرفة النظيفة:فلسفة تطوير برمجيات مبنية على تفادي الأخطاء في البرمجة باتباع عملية فحص صارمة.

عزيزي الدارس أرجو أن لا يغيب عن ذهنك عند قراءة مادة الوحدة محاولة الإجابة عن الأسئلة التقويمية ، والعودة إلى النص للتأكد من صحة الإجابة ، كما أرجو عدم الرجوع إلى إجابات التدريبات إلا بعد محاولة حلها أولاً.

نأمل أن تؤدي دراستك إلى هذه الوحدة إلى إثراء فهمك وتعرفك التحقق والمصادقة على صحة البرمجيات

أهداف الوحدة

الهدف من هذه الوحدة هو إعطاء الدارس، مدخلاً لأعمال التحقــق والمصادقة على صحة البرمجيات مع تركيز خاص لأساليب التحقق الساكنة، وبنهاية دراسة هذه الوحدة يجب أن يكون الــدارس قــادراً على أن :

- يشرح الفرق بين التحقق من البرمجية والمصادقة عليها.
- يصف طبيعة الأخطاء الأكثر شيوعا وانتشارا في البرامج.
- يعدد أهم فحوصات البرامج وكيفية اكتشاف الأخطاء بها.
- يشرح لماذا يعتبر التحليل الساكن الآلي للبرامج أســلوباً مهمــاً وفعالاً للتحقق.
- ينشئ جدول اختبار الصندوق الأسود.
- ينشئ جدول اختبار الصندوق الأبيض.
- يصف اختبار بيتا وفوائده.
- يعدد السمات الأساسية لاختبار الاندماج بنوعيه من أعلى لأسفل ومن أسفل لأعلى.
- يشرح طريقة الغرفة النظيفة لتطوير البرامج ولماذا هي طريقة فعالة وما مميزاتها.
- يميز بين طرق تطوير البرمجيات عن طريق أســاليب التحقــق

توطئة

التحقق والمصادقة هو الاسم المطلق على عمليات الفحص والتحليل التي من شأنها التأكد من أن برامج الحاسب الآلي مطابقة لمواصفاتها وأنها تفي باحتياجات العملاء الذين يدفعون تكاليف هذه البرامج ، وعمليات التحقق والمصادقة هي عمليات مستمرة طوال عمر البرنامج كله ، حيث تبدأ من مرحلة مراجعة المتطلبات وتستمر حتى مراجعات التصميم وفحوصات تحرير نص الشيفرة البرمجية إلى اختبار المنتج، ويجب أن يكون هناك أنشطة تصحيح وتصويب في كل مرحلة من هذه المراحل ، وتقوم هذه النشاطات على فحص نتائج كل مرحلة والتأكد من مطابقتها لما تم تحديده لها ، ويجب التنويه هنا إلى أن التحقق والمصادقة ليسا نفس الشيء رغم أنه قد يتم الخلط بينهما ، وهو ما أعرب عنه بوهيم (1979م) حيث عرف كلاً من التحقق **(Verification)** والمصادقة **(Validation)** كمايلي :

❶التحقق : يعني إنتاج برمجية خالية من الأخطاء.

❷المصادقة : تعني التأكد من تلبية البرنامج لمتطلبات العملاء.

إن البرمجيات غاية في التعقيد ، والأسلوب السائد للتحقق من جودتها ــ في الوقت الراهن ــ هو الاختبار ، والذي يستهلك نسبة هائلة (تصل أحياناً إلى 50%) من المجهود اللازم لتطوير النظام، فعلى سبيل المثال يوجد في شركة ميكروسوفت **(Microsoft)** العديد من المختصين بأمور الاختبار كما يوجد كذلك العديد من المختصين بعمليات البرمجة.

ومن المثير للجدل أن التثبت من سلامة البرنامج يشكل مشكلة كبيرة ونحن في حاجة لأسلوب جيد وفعال لحل تلك المشكلة ، وفي الغالب فإنه عند نهاية المشروع فإن أصعب القرارات الواجب اتخاذها هو القرار بين الاستمرار في عملية الاختبار

أو تسليم البرنامج إلى العملاء الراغبين فيه.

ويشمل دور التصديق التأكد من أن البرنامج مطابق للمواصفات التي وضعت له والتي تشمل (المتطلبات الوظيفية وغير الوظيفية المحددة له) ، بل تذهب أبعد من ذلك إلى التأكد من مطابقة البرنامج لمواصفاته وحتى إلى إثبات أن البرنامج يقوم بعمل ما يتوقع منه العميل مغايرا لما تم تحديده.

والتصديق على متطلبات البرنامج في وقت مبكر ضروري جداً حيث إنه من السهل حدوث أخطاء ومحذوفات في متطلبات البرنامج وفي هذه الحالات فإن البرنامج النهائي لن يكون مطابقاً لما يتوقع العميل منه، وبالرغم من ذلك ففي الواقع لا يتوقع من عملية التصديق على المتطلبات اكتشاف كافة المشاكل الموجودة في المتطلبات، ففي بعض الأحيان تكتشف بعض الأخطاء والنواقص عند اكتمال التنفيذ فقط.

وسنبدأ هذه الوحدة بمناقشة المشكلة العامة للاختبار ، حيث نناقش فيها أعمال التحقق والمصادقة وكيفية التخطيط له ، ثم كيفية عمل فحوصات البرامج بفرعياتها المختلفة والتي تشمل (فرق الفحص ، وعملية الفحص ،و نموذج عملية الفحص ،و قوائم الفحص ، ومعدلات الفحص) ، ثم نعرج بعد ذلك على عدة اختبارات أهمها اختبار التحليل الساكن الآلي ، واختبار الصندوق الأسود ، واختبار الصندوق الأبيض ، واختبار بيتا.

إن مشكلة اختبار أجزاء كبيرة من البرمجيات والتي تتكون من عدة أجزاء متصلة تعد مشكلة صعبة خصوصًا لو أن الأجزاء متصلة معًا ومرتبطة لتكون وحدة واحدة. والبديل هنا هو اختبار التجزيء والذي من أمثلته التطوير من أعلى لأسفل حيث يفتح هذا الاختبار طريقًا أمام المستخدمين لرؤية إصدار نسخة مبكرة من النظام ، ولذلك فهو مفيد جدًا لاختبار الصلاحية.

إن تطوير البرمجيات عن طريق الغرف النظيفة (إعداد برمجيات خالية من الأخطاء بدلا من اكتشافها وتصحيحها) والتي سيتم التعرض له بالتفصيل في نهاية الوحدة تمثل طريقة مكلفة جدا وتحتاج إلى إمكانيات بشرية هائلة ، وكذلك يمكن اقتراح خطط اختبار عملية أخرى ، والتحقق الرسمي من سلامة برنامج ما باستخدام القوانين الرياضية الصارمة للوصول إلى الصحة والدقة والتي تستخدم لعدة أنظمة وبخاصة أنظمة السلامة منها، ولقد أظهرت الدراسات أنه من المفيد للذين يقومون بعملية الاختبار أن يكونوا أناساً آخرين غير المبرمجين ، فعلى سبيل المثال شركة ميكروسوفت توظف عديداً من فرق المبرمجين (الذين يكتبون البرامج) بالإضافة إلى فرق منفصلة تمامًا من المختبرين (الذين يختبرون هذه البرامج).

1. طبيعة الأخطاء (The Nature of Errors)

سوف يكون من الملائم معرفة كيفية ظهور الأخطاء لأننا فيما بعد سنحاول تجنب هذه الأخطاء أثناء كل مراحل التطوير، وبالمثل فإنه من المفيد معرفة أكثر الأخطاء شيوعًا وانتشارًا لأننا فيما بعد سنبحث عنها أثناء التحقق من جودة البرنامج وخلوه من الأخطاء ومن المؤسف أن البيانات غير الشاملة (التامة) هي فقط ممكنة لإعطاء جمل هلامية (غامضة) بخصوص تلك الأشياء.

وتُعد المواصفات مصدراً شائعاً للأخطاء ، فنظام البرنامج له مواصفات عامة مشتقة من تحليل المتطلبات ، بالإضافة إلى أن كل مكون من مكونات البرنامج له مواصفات فردية تخصه مشتقة من التصميم الهندسي ، ومواصفات أي من المكونات قد تكون غامضة (غير واضحة) ، أو غير كاملة (ناقصة) ، أو خاطئة، وأي من تلك المشكلات يجب بالطبع أن تُكتشف وتعالج عن طريق التحقق من جودة المواصفات وخلوها من العيوب قبل عملية تطوير المكون نفسه، لكن

بالطبع فإن هذا التحقق لا يمكن ولن يمكن أن يكون مؤثرًا (فعّالاً) بصورة كاملة ، ولذلك فهناك غالبًا مشاكل بخصوص مواصفات المكونات، هذا ليس كل شيء فهناك مشكلات أخرى بخصوص المواصفات ، أثناء البرمجة فإن مطور المكون قد يسيء فهم مواصفات المكون.

ويوجد نوعٌ آخر من الأخطاء يقع حينما يحوي المكون أخطاء ولا يخضع لمواصفاته وهذا قد يرجع إلى نوعين من المشكلات :

- أخطاء في منطق الشيفرة البرمجية.
- الشيفرة البرمجية لا تغطي كل جوانب المواصفات.

وهذا النوع من الأخطاء يكتشف حين يفشل المبرمج في تقدير وفهم كل تفاصيل المواصفات وبناء على ذلك يقوم بحذف بعضاً من الشيفرة البرمجية الضرورية.

وأخيرًا فإن من أهم أنواع الأخطاء التي قد تظهر عند تشفير (تكويد) أحد المكونات هي :

- عدم إعطاء قيم بدائية للبيانات.
- تكرار الحلقات عدد مرات غير صحيح.
- أخطاء في القيم الحدية.

القيم الحدّية هي قيم البيانات عند أو بالقرب من قيم محددة – فمثلاً لو فرض أن أحد المكونات يجب أن يقرر إذا كان على الشخص التصويت في الانتخابات أم لا استنادًا إلى عمره. فإذا كان سن التصويت هو 18 سنه فإن القيم الحدّية (المتاخمة) بالقرب من القيمة المحددة هي 17، 18، 19.

كما شاهدنا فإن هناك العديد من الأشياء التي يمكن أن تحمل أخطاء وربما لذلك فإنه ليس من المدهش (الغريب) أن التحقق من جودة البرنامج وخلوه من العيوب نشاط يستهلك وقتًا طويلاً.

2. التحقق والمصادقة الساكنة والديناميكية

(Static and Dynamic Verification and Validation) :

تشمل أعمال التحقق والمصادقة عموما طريقتين للفحص والتحليل ، وهما :

①فحص البرنامج **(Software Inspection) :**

ويشمل التحليل والتدقيق على مخططات البرنامج المختلفة مثل مستند المتطلبات، ومخططات التصميم ، وشيفرة البرنامج المصدر ، ويمكن أن تطبق هذه الفحوصات على كافة مراحل العملية، ويمكن أن تعزز هذه الفحوصات بنوع من أنواع التحليل الآلي للمستفيد الأصلي للنظام أو المستندات ذات العلاقة ، ويجب ملاحظة أن الفحوصات والتحليل الآلي للبرنامج هي أساليب ساكنة حيث إنها لا تتطلب أن يتم تشغيل النظام.

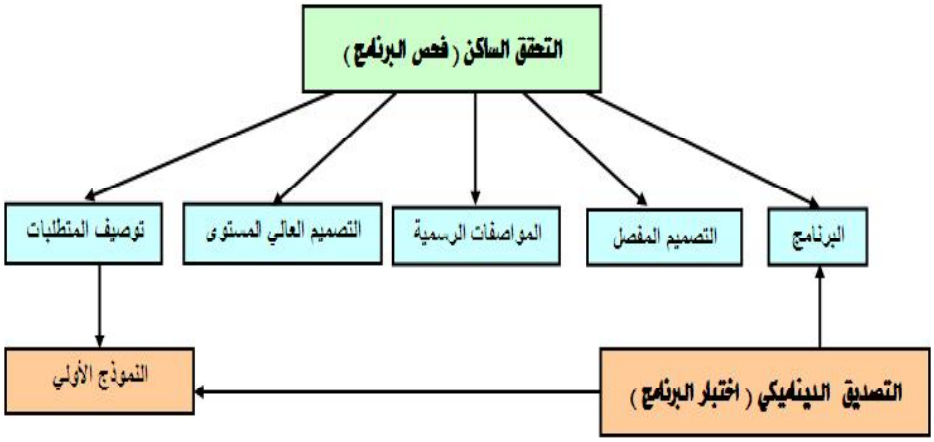
وتتكون تقنيات الفحص من معاينة البرنامج ، والتحليل الآلي للشيفرة البرميجة الأصلية ، والتحقق المنهجي الرسمي ، على أن التقنيات الثابتة تتأكد فقط من وجود توازي بين البرنامج وبين مواصفاته ولا يمكنها التثبت من فائدة البرامج التشغيلية ، كما أن هذه الأساليب لا يمكن لها أن تكشف الخصائص غير التشغيلية للبرنامج مثل الأداء والاعتمادية ، لذلك فأنه من الضروري القيام بنوع آخر من الاختبار للبرنامج ، وبالرغم من أن معاينات البرنامج الآن تتم بشكل واسع إلا أن تجربة البرامج ما زالت هي الأسلوب السائد للتحقق والتصديق ، وذلك لأن التجربة تنطوي على تشغيل البرنامج بمعطيات وبيانات مماثلة للبيانات التي سيقوم البرنامج بمعالجتها في الواقع ، حيث يتم استنباط وجود أية أخطاء أو مشاكل في البرنامج بمعاينة مخرجات البرنامج والتدقيق عن الأشياء غير العادية ، ويمكن القيام بالاختبار خلال مرحلة التنفيذ للتحقق من أن البرنامج يعمل كما أراد له المصمم وذلك بعد الانتهاء من التنفيذ.

ويوضح الشكل 5.1 موضع فحص البرنامج واختباره ضمن عملية النظـــام ، وتوضح الأسهم مراحل العملية التي قد تستخدم فيها هذه التقنيات.

ولذلك فإن فحوصات البرنامج يمكن أن تستخدم في كل مراحـــل تطـــوير البرنامج ولكن في المقابل من ذلك ، فإن الاختبار يمكن تطبيقه عندما يكون هناك نموذج أولي أو برنامج قابل للتنفيذ، وإن شاء الله سوف نتناول فحوصات البـــرامج بصورة مفصلة في الجزء 5.6 من هذه الوحدة.

②اختبار البرنامج **(Software Testing) :**

ويشمل تنفيذ تطبيق البرنامج عن طريق بيانات اختبارية ومعاينة مخرجـــات البرنامج وسلوكها التشغيلي للتأكد من أن البرنامج يفي بما هو مطلوب منه.



الشكل 5.1 : شكل توضيحي لمراحل التحقق والمصادقة الساكنة والديناميكية

ولذلك فإن فحوصات البرنامج يمكن أن تستخدم في كـــل مراحـــل تطـــوير البرنامج ولكن في المقابل من ذلك ، فإن الاختبار يمكن تطبيقه عندما يكون هناك نموذج أولي أو برنامج قابل للتنفيذ. وإن شاء الله سوف نتناول فحوصات البـــرامج بصورة مفصلة في القسم الخامس من هذه الوحدة.

③اختبار البرنامج **(Software Testing) :**

ويشمل تنفيذ تطبيق للبرنامج عن طريق بيانات اختبارية ومعاينة مخرجات البرنامج وسلوكها التشغيلي للتأكد من أن البرنامج يفي بما هو مطلوب منه.

وهناك اختبارات متعددة تستخدم في المراحل المختلفة لإعداد البرمجيةهي :

- اختبار الخلل : **(Defect Testing)** : لاكتشاف الاختلاف بين البرنامج وبين مواصفاته ، حيث يكون السبب في هذه الاختلافات ــ في أكثر الأحيان ــ لخلل في البرنامج نفسه فتكون التجارب مصممة لإظهار الخلل في النظام بدلاً من محاكاة استخدامه التشغيلي.
- الاختبار الإحصائي : **(Statistical Testing)** : ويستخدم لاختبار أداء البرنامج والاعتماد عليه ولكشف طريقة عمله تحت الظروف التشغيلية المختلفة، وهذه الاختبارات مصممة على أن تعكس مدخلات المستخدم العادي وتكرارها ، ويمكن بعد تشغيل البرنامج تقدير اعتمادية البرنامج ، وذلك باحتساب عدد مرات فشل النظام ، ويمكن الحكم على أداء البرنامج بقياس وقت التنفيذ ووقت استجابة البرنامج بينما يقوم بمعالجة البيانات التجريبية.

وبطبيعة الحال فإنه لا توجد حدود ثابتة وسريعة بين أساليب الاختبار هذه ، فخلال اختبار الأعطال يحصل المختبرون على استشعار الاعتمادية ، ومن خلال الاختبار الإحصائي من المتوقع أن يتم اكتشاف بعض الأعطال، لأن الهدف النهائي من عملية التحقق والمصادقة هو العمل على تأسيس الثقة بأن البرنامج صالح للغرض الذي طور من أجله ، وهذا لا يعني أن البرنامج سيكون خالياً من الأخطاء ، ولكن معناه أن البرنامج جيد للاستخدام المرجو منه.

ويعتمد مستوى الثقة المطلوب للتحقق والمصادقة على كل من الأهداف المرجوة من البرنامج ، وتوقعات مستخدمي البرنامج ، والبيئة التسويقية الحالية لهذا البرنامج والتي يمكن توضيحها بالتفصيل كالتالي :

1) وظيفة البرنامج **(Software Function)** : يعتمد مستوى الاعتمادية المطلوب بأهمية البرنامج للمؤسسة فالاعتمادية المطلوبة لبرنامج للتحكم بمستوى نظام أمنى هو أعلى بكثير من الاعتمادية المطلوبة لبرنامج تجريبي تم تطويره لعرض فكرة جديدة.

2) توقعات المستخدمين **(User Expectations)** : إنه لمؤسف في صناعة البرمجيات أن تتكون لدى الكثير من المستخدمين توقعات متدنية عن البرامج التي يتم تطويرها ولا يكونو مندهشين عندما تفشل هذه البرامج في التشغيل، ويكون المستخدمون مستعدين لقبول هذه الإخفاقات عندما تفوق فوائد استخدام البرامج إخفاقاته، وبالرغم من ذلك فقد تضاءل تحمل المستخدمين لفشل البرامج مما كان عليه في التسعينات، والآن فإن تسليم أنظمة غير موثوق بها أصبح أقل قبولاً، لذلك يجب على شركات تطوير البرمجيات استغراق وقت أطول في أعمال التحقق والمصادقة.

3) البيئة التسويقية **(Marketing Environments)** : عندما يتم تسويق برنامج ، يجب أن يأخذ البائعين في اعتبارهم البرامج المنافسة ، والأسعار التي يود المستخدم أن يدفعها مقابل هذا النظام ، والجدول المطلوب لتسليم هذا النظام. وعند ما يكون هناك منافسون قليلون في السوق لشركة ما فإن هذه الشركة قد تود إصدار برنامج غير مصحح بشكله النهائي لأنها تريد أن تكون أول الداخلين به إلى السوق، وفي الحالات التي يكون فيها المستخدمون غير مستعدين لدفع مبالغ كبيرة على البرنامج ، فإنهم سيكونون حينئذ مستعدين لقبول بعض الأخطاء في هذه البرامج ، ويجب أن تؤخذ جميع هذه العوامل في الحسبان.

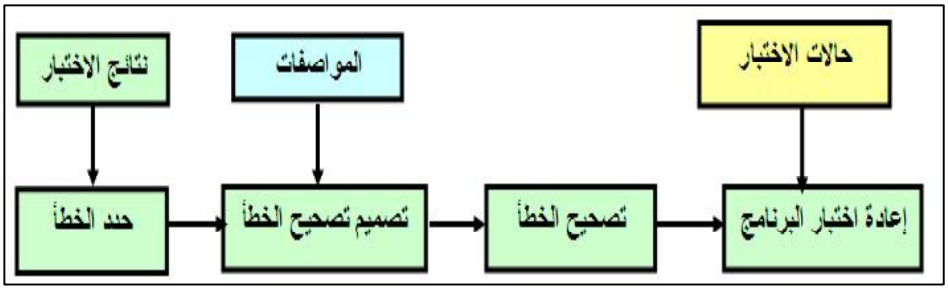
أسئلة تقويم ذاتي

1. صنف أهم الأخطاء شيوعا وانتشارا.
2. لماذا ليس من الضرورى أن يكون البرنامج خاليا تماما من العيوب قبل أن يصل للعملاء؟

3. اكتشاف وتصحيح الأخطاء

(Testing and Debugging)

في العادة يتم اكتشاف الأخطاء الموجودة في برنامج ما حيث يتم تصحيح الأخطاء ويجب عندئذ تعديل البرنامج لتصحيح هذه الأخطاء ، وعملية التصحيح تدمج دائما مع أعمال التحقق والمصادقة الأخرى إلا أن الاختبار أو بصفة عامة التحقق والمصادقة والتصحيح هي عمليات مختلفة لا يجب أن يتم دمجها ، حيث إن التحقق والمصادقة هي عملية اكتشاف وجود أخطاء في البرنامج ، أما عملية التصحيح ، كما هو واضح في شكل 5.2 ، فهي العملية التي تحدد مكان هذه الأخطاء حيث يتم تصحيحها.



شكل 5.2 : شكل توضيحي لعملية تصحيح الأخطاء

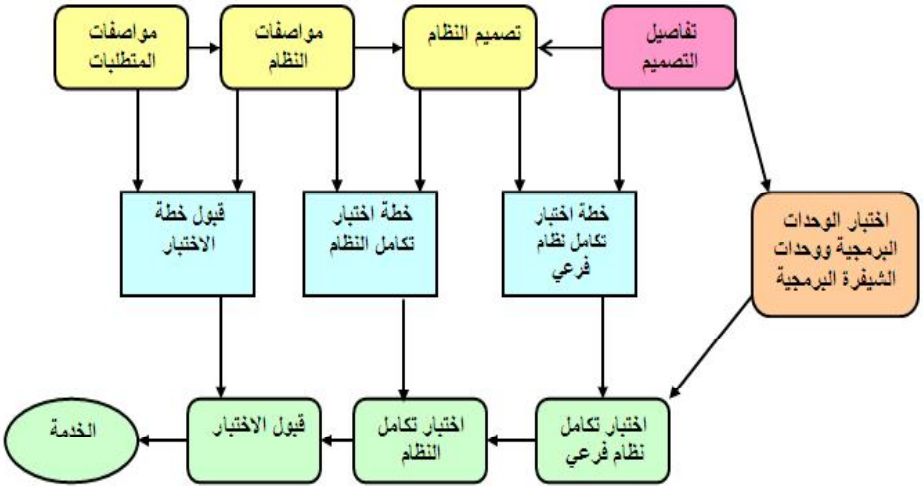
ولا توجد طريقة سهلة لتصحيح البرنامج حيث يتفحص المهندسون المهرة كلاً من نوع الخطأ ،و لغة البرمجة ، وعملية البرمجة لتحديد موقع الخطأ ، وهذا مهم جداً حيث تعود المبرمجون على التعرف على أخطاء البرمجة الشائعة مثل عدم زيادة العداد ومطابقة هذه الأخطاء بالأنماط المكتشفة ، كما يمكن اعتبار خصائص أخطاء لغات البرمجة مثل التوجه الخاطئ الذي يحصل في لغة سي، كما أن إيجاد الخطأ في برنامج ما ليس دائماً سهلاً لأن الخطأ ليس بالضرورة قريباً للنقطة التي يفشل فيها البرنامج ، ولإيجاد الخطأ في برنامج ما فإن مبرمج هذه البرنامج قد يضطر إلى عمل اختبارات إضافية تساعد على معرفة مصدر الخطأ في البرنامج ، وقد يكون التتبع اليدوي للخطأ وبتشغيل التنفيذ ضرورياً وفي بعض الحالات فإن الأدوات المستخدمة لتصحيح الأخطاء التي تجمع المعلومات عن طريقة تنفيذ البرنامج مفيدة.

4. التخطيط للتحقق والمصادقة (Verification and Validation Planning) :

التحقق والمصادقة عملية مكلفة ، فبالنسبة لبعض الأنظمة الكبيرة مثل أنظمة الوقت المباشر **(Online-Systems)** التي لها أهداف غير وظيفية معقدة ، فإن نصف تكاليف التطويرتقريباً تصرف على التخطيط بهدف الحصول على الفائدة القصوى من هذه الفحوصات والاختبارات وبهدف السيطرة على تكاليف عملية التحقق والمصادقة.

- **نموذج الاختبار والتطوير (V-Model Development) :**

يجب أن يبدأ التخطيط للتحقق والمصادقة في وقت مبكر من عملية تطوير البرنامج ، ويوضح النموذج في الشكل 5.3 كيفية استخراج خطط الاختبار من مواصفات النظام وفق تصاميمه ، ويدعى هذا النموذج في بعض الأحيان نموذج في (V) (أقلب شكل 5.3 على جانبية لرؤية الـV) .



شكل 5.3 : شكل توضيحي لنموذج "V" للاختبار والتطوير

ويوضح هذا الشكل أيضاً كيفية تقسيم أنشطة التحقق والمصادقة إلى عدد من المراحل بحيث تقاد كل مرحلة من الاختبارات التي حددت لها لاكتشاف مطابقة البرنامج لمواصفاته وتصميمه ، ويجب أن تشمل عملية التخطيط للتحقق والمصادقة التوازن بين كل من الأساليب الديناميكية والثابتة ، وكذلك أن توضح المعايير والإجراءات الخاصة بفحوصات واختبارات البرامج وإعداد الكشوفات التي بواسطتها يتم القيام بهذه الفحوصات وأن dتم تحديد خطة الاختبار ، ويعتمد الجهد النسبي المخصص للفحوصات والاختبارات على نوع النظام الذي يتم تطويره والخبرة السابقة للمنشأة في هذا المجال فكلما كانت أهمية النظام كبيرة كلما كان ضرورياً تخصيص جهد إضافي أكبر للتقنيات الثابتة.

إن التخطيط للاختبار يتضمن وضع معايير لعملية الاختبار بدلاً من شرح اختبارات المنتج ، ذلك لأن خطط الاختبار ليست فقط مستندات إدارية لأنها مصممة لتستخدم من قبل مهندسي البرامج القائمين على تصميم وإجراء هذه الاختبارات ، كما أن هذه الخطط تسمح للموظفين الفنيين بأن تتكون لديهم فكرة

عن الصورة الكلية لاختبارات النظام مما يسمح بوضع عملهم في هذا السياق ، كما توفر هذه الاختبارات للقائمين على المشروع المعلومات التي تضمن توفر موارد الأجهزة والبرامج المناسبة لفريق الاختبار.

- **هيكل خطة اختبار البرمجيات (Structure of a Software Test Plan) :**

يجب أن تأخذ هذه الخطة في الاعتبار كميات كبيرة من الأعمال المضاعفة (حالات الطوارئ) بحيث يتم الإحاطة الكاملة بالتأخيرات الخاصة على التصاميم والتنفيذ ويتم تحويل الموظفين الذين تم تخصيصهم لعمليات الاختبار إلى أنشطة أخرى ، وقد قام الباحث هاتون (1986م) بشرح جيد لخطط الاختبارات وعلاقتها بخطط الجودة الأكثر عمومية، وكما هو الحال للخطط الأخرى فإن خطة الاختبار ليست مستندا ثابتاً ويجب أن تتم مراجعتها بانتظام ، ذلك لأن الاختبارات أنشطة معتمدة على استكمال التنفيذ فإذا كان جزء من البرنامج غير مكتملاً فإنه لا يمكن القيام بالاختبار المتكامل ، والجدول رقم (1) يوضح العناصر الأساسية لخطة اختبار برمجية.

جدول رقم (1) يبين بنية خطة اختبار البرمجية (Structure of a Software Test Plan)

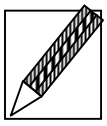
الفرعية	وصفها
عملية الاختبار The Testing Process	وتشمل وصف للمراحل الرئيسية لعملية الاختبار.
تعقب المتطلبات Requirements Traceability	إن ما يهم المستخدمين هو أن يفي البرنامج بما هو مطلوب منه ويجب أن يتم التخطيط للاختبار بحيث يتم اختبار كافة المتطلبات بشكل مستقل.
البنود المختبرة Tested Items	يجب أن تحدد بنود العملية للبرمجية التي يجب أن تختبر.
جدولة الاختبارات Testing Schedule	جدول كلي للاختبارات وتخصيص الموارد لهذا الجدول ومن الواضح أن هذا مرتبط بالجدول الشامل لتطوير المشروع الأكثر عموماً.

الفرعية	وصفها
إجراءات تدوين الاختبارات Test recording Procedures	أنه غير كافٍ أن يتم إجراء الاختبارات فقط ، يجب أن تدون نتائج الاختبارات بشكل منظم وبنحو يسمح بالتدقيق في عملية الاختبار للتأكد من أنه تم القيام بها بشكل سليم.
متطلبات الأجهزة والبرمجيات Hardware and Software Requirements	يجب أن يحدد في هذا الجزء البرمجيات اللازمة وعتاد الحاسب المطلوب.
القيود Constrains	يجب أن يتم في هذه الفرعية التوقع للقيود التي تؤثر على عملية الاختبار مثل قلة عدد الموظفين.

5. فحوصات البرامج (Program Inspections)

فحوصات البرامج هي مراجعات هدفها اكتشاف الأخطاء والتي قد تكون موجودة في البرنامج ، وقد تم تطوير فكرة القيام بفحص رسمي أول مرة في شركة **(IBM)** في السبعينات وجرى شرحها من قبل الباحث فاجان (1979م) حيث إنها الآن الطريقة المستخدمة بشكل واسع للتحقق ، ومنذ ذلك الحين تم تطوير العديد من البدائل للطريقة الأصلية المطورة من قبل فاجان ، **وبرغم ذلك فإن كل هذه الطرق مبنية على الفكرة الأصلية لفاجان والمتمثلة في أن يقوم فريق مكون من أفراد بتخصصات مختلفة بالقيام بمراجعة الشيفرة البرمجية المصدر سطراً سطراً.**

تدريب (1)



ما هو الفرق الأساسي بين فحوصات البرنامج وبين الأنواع الأخرى من مراجعة الجودة.

- **فرق التفتيش (الفحص) (Inspection Teams) :**

إن عملية الفحص هي عملية ذات طابع رسمي ويقوم بها في العادة فريق مكون من أربعة أشخاص حيث يقوم أعضاء الفريق بتحليل الشيفرة البرمجية والكشف عن أية أخطاء فيه (إن وجدت) ، وقد اقترح الباحث فاجان توزيع الأدوار كمايلي : كاتب (مبرمج الشيفرة) ، وقارئ ، وفاحص ، ومنسق ، وذلك على أعضاء الفريق بحيث يقوم القارئ بقراءة نص البرنامج بصوت عالٍ لأعضاء الفريق ، ويقوم الفاحص بإجراء الاختبارات من منظور اختباري (يجد الأخطاء والنواقص والمتعارضات) ويقوم المنسق بالترتيب للعملية (يرأس الجلسات ويديرها ، ويدون الأخطاء المكتشفة) وبفعل تطور خبرة المنشآت في مجال الفحوصات فقد ظهرت اقتراحات أخرى لتشكيل الفريق ، ففي مناقشة لإدخال عملية التطوير الناجحة لشركة HP قام الباحثان جراوي وفان إيلاك (1994م) باقتراح ستة أدوار لأعضاء الفريق كما هو موضح في الجدول رقم (2) ، حيث يمكن أن يأخذ العضو الواحد أكثر من دور واحد في الفريق ، وكذلك حجم الفريق قد يكون مختلفاً من فحص إلى أخر.

الجدول رقم (2) : مكونات أعضاء فريق الفحص السداسي

الدور	الوصف
المؤلف أو المبرمج	**المبرمج أو المصمم المسئول عن إنتاج البرنامج أو المستند ، مسئول عن تصحيح الأعطال المكتشفة خلال عملية الفحص.**
الفاحص	**يكتشف الأخطاء والمحذوفات وعدم الانسجام في البرامج والمستندات.**
القارئ	**يعيد صياغة الكود أو المستند في جلسة اجتماعات الفحص.**
المسجل(الكاتب)	**وهو الذي يسجل نتائج مقابلة الفحص.**
رئيس الفريق أو المنسق	**يدير العملية ويعمل على تسهيل الفحوصات، ويرسل تقارير عن نتائج العملية إلى كبير المنسقين.**
كبير المنسقين	**مسئول عن إجراء تحسينات وتحديث الكشوفات وتطوير المعايير.**

وقد لاحظ جرادي وفان إيلاك إن هناك حاجة دائماً لدور القارئ وفي هذا الصدد قاما بتعديل الاقتراح الأصلي لفاجان والذي كان فيه جزء مهم من العملية متعلقاً بقراءة النص بصوت عالي، كما أن جيب وجراهام (1993م) لا يريان ضرورة لوجود قارئ ويقترحان أن يتم اختيار فريق الفحص بشكل يعكس اختلافات وجهات النظر (مثل مختبر مستخدم إدارة جودة ...الخ).

- الشروط المسبقة للتفتيش **(Inspection Pre-condition) :**

وقبل أن يبدأ الفحص للبرنامج فإنه من الضروري :

①أن يكون هناك مواصفات دقيقة.

②أن يكون أعضاء فريق الفحص على دراية بمعايير المؤسسة.

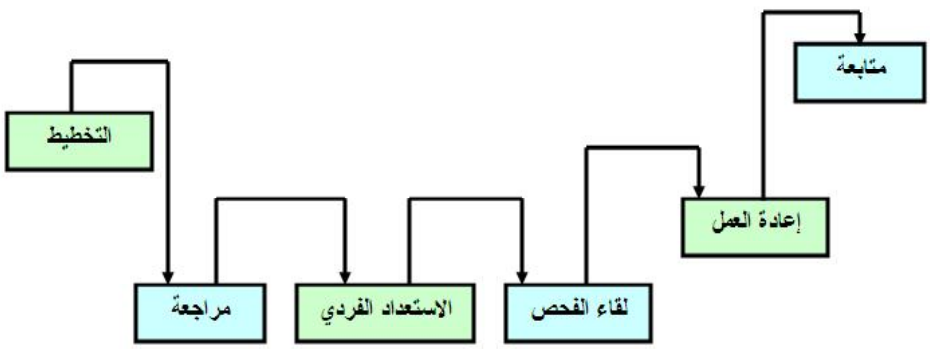
③أن يكون هناك نسخة صحيحة من حيث قواعد اللغة وحديثة من البرنامج الذي يراد فحصه.

④يجب أن تتوقع الإدارة أن عملية الفحص ستزيد من التكلفة في بداية عملية إعداد البرنامج.

⑤يجب أن لا تستخدم الإدارة عملية الفحص لتقويم العاملين.

- عملية التفتيش (الفحص) **(The Inspection Process)** :

وهناك نموذج عام لعملية الفحص موضح في الشكل 5.4 ، حيث يمكن الاقتباس وتكييف هذا النموذج وفق متطلبات المنشأة المعنية ، ويكون المنسق مسئولاً عن التخطيط لعملية الفحص ، وهذا يتضمن اختيار أعضاء فريق الفحص والترتيب لغرفة الاجتماعات، وأن يتأكد من اكتمال المواد المراد فحصها والتأكد من اكتمال المواصفة الخاصة به ، كما يتم تقديم البرنامج المراد فحصه إلى الفريق خلال مرحلة المراجعة ويقوم مؤلف البرنامج بوصف ما يجب أن يقوم به البرنامج وتلي هذه المرحلة مدى استعداد الأفراد بحيث يقوم كل عضو من أعضاء الفريق بدراسة المواصفة والبرنامج.



شكل 5.4 : شكل توضيحى لعملية الفحص

ويجب أن يكون الفحص ذاته قصيراً نسبيا (لا يزيد عن ساعتين) ويجب أن يكون فحصاً بصفة كاملة لاكتشاف الأخطاء الشواذ وعدم المطابقة للمواصفة والمعايير ، ويجب الاَ يقوم فريق الفحص بتقديم أي اقتراح حول كيفية تصحيح هذه الأخطاء أو الفرضية على تغير العناصر.

وبعد إجراء الفحص يتم تعديل البرنامج وتصحيحه من قبل مؤلف ومن ثم يقرر المنسق ما إذا كان إجراء فحص آخر ضرورياً ، وفي المقابل فقد يقرر أنه لا توجد ضرورة لإجراء فحص كامل مرة أخرى وأَنَ المشاكل قد تم تصحيحها ويقوم بعد ذلك بالموافقة على إصدار البرنامج.

ويجب أن يتم القيام بعمل كشف للأخطاء الشائعة في البرامج وذلك بالتنسيق والاستعانة بالموظفين ذوي الخبرة ، وأن يتم تحديث هذه الكشوفات كلما زادت الخبرة في عملية الفحص ، كما يجب طبع كشوف فحص مختلفة لكل نوع من لغات البرمجة ، وتختلف هذه الكشوف من لغة برمجة إلى أخر؛ وذلك لأن كل لغة برمجة لديها مستوى مختلف من مستويات الكشف عن الأخطاء ، فمثلاً برنامج مكتوب بلغة آدا (Ada) يتم الكشف عن عدد العناصر كاملة بينما لا يقوم برنامج سي بذلك.

- **قوائم الفحص (Inspection Checklists)** :

ينصح جيلب وجراهام (1993م) أن تقوم كل منشأة بتطوير ووضع الكشوف الخاصة بها ويجب أن تكون هذه الكشوفات مبنية على المعايير والممارسات المحلية ولابد أن يتم تحديثها بشكل دوري كلما ظهرت أنواع أخرى من الأخطاء. والجدول رقم (3) يوضح المعاينات (الفحصوات المطلوبة) التي يمكن إِجراؤها خلال مرحلة الفحص.

الجدول رقم (3) : يشمل قائمة الفحوصات التي يمكن إجراؤها خلال مراحل الفحص

فئة الخطأ (Fault Class)	الفحص المطلوب (Inspection Check)
أخطاء بيانات Data Faults	• هل كل متغيرات البرنامج تم وضع قيمة بدائية لها قبل استخدامها؟ • هل كل الثوابت تم تسميتها؟ • هل الحد العلوي للمصفوفة يساوي حجم المصفوفة أم هو أقل ؟ • هل هناك احتمال ظهور أرقام أعلى من الحد المسموح به؟
أخطاء تحكم Control Faults	• لكل جملة شرط هل الحالة صحيحة؟ • هل لكل مسار تأكيد لانتهائه؟ • هل عدد الأقواس في الجمل المركبة سليم؟
أخطاء دخل/ خرج Input/Output Faults	• هل كل المتغيرات الداخله مستخدمة؟ • هل كل متغيرات الخرج لها قيمة قبل إخراجها؟ • هل المدخلات غير متوقعة تسبب مشاكل؟
خطأ واجهة Interface Faults	• هل كل دالة وطريقة استدعاء لها العدد الصحيح من المتغيرات؟ • هل هناك توافق في أنواع المتغيرات؟ • هل المتغيرات في الترتيب الصحيح؟ • لو كانت المتغيرات متشاركة في الذاكرة فهل لها نفس موديل المشاركة في الذاكرة؟
أخطاء تخزين Storage Management Faults	• إذا حدث تغيير في تركيبة وصله هل تم إجراء تغيرات صحيحة لكل الوصلات؟ • لو تم استعمال ذاكرة ديناميكية هل تم تخصيصها بشكل صحيح؟ • هل يتم إلغاء تخصيص بوضوح بعد ألا يكون مطلوباً؟
أخطاء استثنائية إدارية Exception Management Faults	• هل كل احتمالات حالات الأخطاء أخذت في الاعتبار؟

وكلما اكتسبت المنشأة خبرة إضافية في عملية الفحص فإنه يمكن الاستفادة من نتائج هذه الفحوص كوسيلة للتحسين ، كما يمكن تحليل الأخطاء التي تم اكتشافها في هذه المرحلة وقد يكون بمقدور فريق الفحص ومطور البرنامج شرح أسباب حدوث هذه الأخطاء ، ويجب العمل على تعديل العملية بشكل لا تتكرر معه هذه الأخطاء كلما أمكن ذلك.

وقد لاحظ الباحثان جيلب وجراهام أن عدداً من المنشآت قد تخلت عن اختبار الوحدة لصالح الفحوصات حيث اكتشفت هذه المنشآت أن الفحوصات كانت فعالة لاكتشاف الأخطاء مما جعل التكاليف المتعلقة باختبار الوحدة غير مبرر ، وكما سيتم مناقشته في وقت لاحق من هذا الفصل فإن فحوصات البرامج قد استبدلت اختبار الوحدة في عملية تطوير برامج الحاسب الآلي باستخدام الغرف النظيفة.

- معدلات التفتيش **(Inspection Rates)** :

وقد لوحظ أن كمية الشيفرة البرمجية التي يمكن فحصها في زمن محدد يعتمد على خبرة فريق الفحص الذي يقوم بهذا العمل ، وعلى لغة البرمجة المستخدمة ، وعلى التطبيق الرئيسي ، وخلال قياس عملية الفحص لدى شركة **(IBM)** وجدت معدلات الفحص التالية :

- يمكن مراجعة 500 عبارة من عبارات نص الشيفرة البرمجية في الساعة خلال مرحلة المراجعة.
- يمكن خلال الإعداد الفردي تنفيذ حوالي 125 عبارة من نص الشيفرة البرمجية في الساعة.
- يمكن فحص حوالي 90 إلى 125 عبارة من نص الشيفرة في الساعة خلال الاجتماع.
- تكلفة فحص 500 سطر في الشيفرة البرمجية يعادل مجهود أربعين رجل – ساعة **(40 Man-Hours)**.

وقد تم التأكيد على هذه الأرقام بواسطة بيانات تم جمعها من شركة (AT&T) (برنارد وبرايس 1994م) حيث أظهر قياس أعمال الفحص نتائج مماثلة.

ويقترح الباحث فاجان أن يكون الوقت المستغرق في الفحص حوالي الساعتين ، وذلك لأن كفاءة اكتشاف الأخطاء تهبط بعد هذا الوقت ، لذلك فمن المفروض أن تكون أعمال الفحص عمليات متكررة وتجري على مكونات صغيرة كل مرة خلال مرحلة تطوير البرنامج،وبقيام فريق فحص مكون من أربعة أشخاص وجد أن تكاليف فحص مائة سطر من الشيفرة البرمجية يعادل تقريباً شخصاً واحداً ليوم واحد ، وهذا يعني أن الفحص ذاته يستغرق حوالي ساعة واحدة وأن كل فرد في الفريق يجب أن يأخذ حوالي ساعة إلى ساعتين للإعداد والتفحص ، وفي المقابل فإن تكاليف اختبار البرامج متغايرة وتعتمد على عدد الأخطاء في البرنامج ، وبرغم ذلك كان الجهد المطلوب للفحص أقل من نصف الجهد الذي يستلزمه الاختبار.

ومن ناحية أخرى يجب أن يتأكد المديرون من أن هناك فاصلاً واضحاً بين أعمال الفحص وبين ممارسات التقويم الخاصة بالأفراد ، ولا يجب أن تستخدم نتائج الفحوصات التي تظهر أخطاء معياراً لتقويم المهندسين ، ويجب أن يكون القائمون على الفرق بشكل على توفير الدعم عند اكتشاف الأخطاء ولا يكون فيها أي معاتبة تتعلق بهذه الأخطاء.

6. التحليل الآلي الساكن (Automated Static Analysis)

إن محللات البرامج (Programs Analyzers) الساكنة هي أدوات برمجية تمر على جميع محتويات نص البرنامج المصدر وتكشف الأخطاء التي فيه والكائنات الشاذة به ولا تتطلب هذه المحللات تشغيل البرنامج ولكنها تتخلل نص البرمجية ومن ثم تحدد الأنواع الخاطئة في عباراتها، وتقوم هذه المحللات باكتشاف ما إذا كانت هذه العبارات قد أُسست بشكل سليم أنها تقوم بالاستنتاجات عن مسار التحكم في البرنامج ، وفي الكثير من الحالات تقوم بحساب كافة مجموعة القيم لبيانات البرنامج ، كما تقوم بالإعداد لإمكانيات اكتشاف الأخطاء التي يقدمها مترجم اللغة.

- فحوصات التحليل الساكن **(Static Analysis Checks)** :

إن القصد من التحاليل الآلية الساكنة هو جذب انتباه الشخص القائم على التحقق للكائنات الشاذة في البرنامج مثل استخدام متغيرات لم يتم استهلالها ، أو متغيرات لم يتم استخدامها ، أو بيانات قد تفوق الحدود ، وقد تم تبيان بعض الأخطاء التي يمكن اكتشافها بواسطة المحللات الساكنة في الجدول رقم (4). ومع أن هذه ليست بالضرورة حالات خطأ فإنه في الكثير من الأحيان تكون هذه الكائنات الشاذة هي نتيجة أخطاء برمجية. ويمكن القول إن التحاليل الآلية الساكنة تفيد لمساعدة عملية الفحص وتعد مساندا لها وليس بديلا عنها.

جدول رقم (4) : ويشمل فحوصات التحليل الآلي الساكن (Static Analysis Checks)

فئة الخطأ Fault Class	فحص التحليل الساكن Static Analysis Check
أخطاء البيانات Data Faults	• متغيرات مستخدمة قبل إجراء الاستهلال. • متغيرات تم الإعلان عنها مصرحة ولكن غير مستخدمة. • متغيرات تم الإعلان عنها مرتين ولكن لم تستخدم بين المهام. • تجاوزات لحدود المصفوفات. • متغيرات غير معلن عنها.
أخطاء في التحكم Control Faults	• كود لا يمكن الوصول إليه. • تفرع غير شرطي إلى حلقات.
أخطاء دخل / خرج Input/Output Faults	• أنتجت المتغيرات مرتين بدون تدخل مهمة
أخطاء في الواجهة البينية Interface Faults	• عدم توافق نوع المتغيرات. • عدم توافق عدد المتغيرات. • عدم استخدام الدوال. • وظائف أو إجراءات لا يتم استدعاؤها.
أخطاء إدارة الذاكرة Storage Management Faults	• مؤشرات غير مخصصة. • حسابات المؤشر.

- مراحل التحليل الساكن **(Stages of Static Analysis)** :

وتشمل مراحل التحليل الساكن :

① تحليل مسار التحكم **(Control Flow Analysis)** : يتم في هذه المرحلة تحديد الحلقات متعددة نقاط الدخول والخروج والشيفرة البرمجية التي لا يمكن الوصول إليها ، وهي شيفرة محاطة بعبارات تحويل **(GOTO)** أو جزء من عبارة يكون شرط التحكم فيه لا يمكن أن يكون صحيحاً.

② تحليل استخدام البيانات **(Data Base Analysis)** :هذه المرحلة تلقي الضوء على كيفية استخدام المتغيرات في البرنامج حيث يدقق على المتغيرات التي لم يجرى لها استهلاك سابقا (قيم بدائية) ، والمتغيرات المكتوبة مرتين دون أن يكون بينها تعيين ، والمتغيرات المصرح بها والمعلن عنها ولكنها لا تستخدم في البرنامج ، كما أن تحليل استخدام البيانات يكتشف أيضاً الاختبارات غير الفعالة حيث تكون حالة الاختبار زائدة ، علماً بأن الحالات المكررة هي حالات لا تتغير فيها القيمة وهي دائماً أما صحيحة أو غير صحيحة.

③ تحليل الواجهة **(Interface Analysis)** : يكشف هذا التحليل توافق الروتين والإعلان عن المتغيرات فيه واستخداماتها ، وهذا التحليل غير ضروري إذا ما استخدمت لغات قوية مثل لغة الجافا ، حيث يقوم مترجم هذه اللغات بهذا التحليل ، ويمكن أن تكتشف تحاليل الواجهات الأخطاء المتعلقة في اللغات ضعيفة المستوى مثل لغة الفورتران ولغة سي ، كما يمكن لهذا التحليل اكتشاف الدوال والبرامج الفرعية والإجراءات المصرحة ولا تستدعي من قبل البرنامج أو نتائج هذه الوظائف التي لا يتم استخدامها.

④ تحليل تدفق المعلومات **(Information Flow Analysis)** : يتم في هذه المرحلة تحديد العلاقة بين متغيرات الإدخال ومتغيرات الإخراج ، ورغم عدم الكشف عن الأشياء الشاذة إلا أن القيم المستخدمة في البرنامج توضح بشكل واضح في قوائم وبذلك يمكن اكتشاف الأخطاء خلال مراجعة الشيفرة البرمجية أو في مرحلة الفحوصات ، ويمكن أن توضح تحاليل مسار المعلومات الشروط التي تؤثر على قيمة متغير معين.

⑤ تحليل المسار **(Path Analysis)** : هذه المرحلة الخاصة بتحليل المضمون تحدد كل المسارات المحتملة في البرنامج وتحدد العبارات المنفذة في كل مسار ، حيث يوضح هذا التحليل نمط التحكم في البرنامج.

إن تحليل تدفق المعلومات وتحليل المسار ينتجان كمية كبيرة من المعلومات ، على أن هذه المعلومات لا تلقى الضوء على الحالات الشاذة ولكنها تقدم البرنامج من وجهة نظر أخرى ، وبسبب حجم المعلومات الكبير الذي ينتج فيه فإن هذه المراحل من التحليل الساكن ربما تستبعد من العملية حيث تستخدم فقط المراحل الأولى التي تكشف الحالات الشاذة بشكل مباشر.

- **استخدامات التحليل الساكن (Uses of Static Analysis) :**

المحللات الساكنة ذات فائدة كبيرة عندما تستخدم لغات مثل سي التي لا تتمتع بتنويع قوي والتدقيق الذي يمكن أن يقوم به مترجم لغة سي المحدود ، لذلك فهناك احتمالات كبيرة لوجود أخطاء في البرنامج يمكن اكتشافها بواسطة إدارة التحليل آلياً ، وهذا ضروري جدا بصفة خاصة عندما تكون لغة مثل سي (و سي ++) مستخدمة في تطوير برامج حساسة جداً ، وتحتوي أنظمة يونيكس ولينكس على محللات ساكنة تدعي(لينت) خاصة بالبرامج المكتوبة بلغة سي ، وتقدم لينت تدقيقاً ساكناً موازياً يمكن أن يقدمه مترجم لغة قوية التنويع مثل لغة جافا.

ولا يمكن أن تُستبدل الأدوات الآلية بالمحللات الآلية الساكنة في عملية الفحص لان هناك بعض أنواع الأخطاء التي لا تكتشف بواسطة المحللات الساكنة ، فمثلاً يمكن لهذه المحللات أن يكشف عن المتغيرات التي لم يرسل أمر استهلاك ولكنها لا تكشف عن عدم صحة أمر الاستهلاك ، وفي اللغات ضعيفة التنويع مثل سي ، تقوم المحللات الساكنة باكتشاف الدالات التي يوجد بها أرقام خاطئة وأنواع الأوامر بها خاطئة ايضاً ولكنها لا تكشف عن الأمر الخاطئ.

ولا يوجد شك في أن المحللات الساكنة للغات مثل سي هي أسلوب فعال لاكتشاف أخطاء البرنامج حيث تبقى بعض الأخطاء اللغوية غير معروفة بواسطة المترجم ، ولكن رغم ذلك فإن مصممي اللغة وللغات الحديثة مثل جافا قاموا

باستبعاد بعض الميزات التي كانت موضع الأخطاء في هذه اللغات حيث يتم إرسال أوامر الاستهلاك لكل المتغيرات ، ولا توجد عبارات **(GOTO)** مما يقلل من فرص القيام بالأخطاء ، كما أن إدارة الذاكرة تتم بطريقة آلية ، وهذا المنهج الخاص بتجنب الأخطاء بدلاً من اكتشافها هو الأسلوب الأكثر فعالية في تحسين اعتمادية البرنامج لذلك المحللات الساكنة المتممة قد لا تكون مناسبة لبرامج جافا.

7. اختبار الصندوق الأسود

(Black Box (Functional)Testing)

اختبار الصندوق الأسود هو ابتكار عينة من البيانات تنوب عن (تمثل) كل البيانات المحتملة (استخدام كل البيانات المحتملة في الاختبار يطلق عليه الاختبار المجهد أو الشاق) ثم نجري البرنامج وندخل البيانات ونرى ما سيحدث. واختبار الصندوق الأسود مزعوم لأنه لا معرفة له بأعمال البرنامج تستخدم كجزء من الاختبار. نحن فقط نفكر في المدخلات والمخرجات ، والبرنامج نفكر فيه على أنه غير مرئي داخل صندوق أسود.و اختبار الصندوق الأسود يسمى أيضًا الاختبار الوظيفي (اختبار المهام) لأنه يستخدم فقط المعرفة بمهام البرنامج.و كمثال فإننا سوف نفكر في إجراء اختبار ينفذ عملية الضرب باستخدام اختبار الصندوق الأسود. مواصفات الإجراء أنه يضرب قيمتين ويعيد منتج الرقمين.

نحن نحتاج إلى بعض البيانات التي تمثل الكل ،فقد تعتقد أن أي قيمة تماثل أي قيمة أخرى لذلك كل ما نحتاج عمله هو اختيار أي قيمتين لتمثيل كل القيم الممكنة وهكذا فإننا نختار مجموعة واحدة من بيانات الاختبار.

رقم الاختبار	البيانات	النتيجة المتوقعة
1	8 ، 12	96

على سبيل الحذر قد نشعر بالقلق لأن هذا محدود جدًا، مدركين أن الأرقام السالبة مختلفة نوعًا ما عن الأرقام الموجبة، فاختر ممثلاً عن الأعداد السالبة

ونستخدم كل الدمج.

رقم الاختبار	البيانات	النتيجة المتوقعة
1	12 ، 8	96
2	8 ، 7-	56-
3	20- ،12	240 -

مازال الاختبار غير كامل نسبيًا فقد نجد أن الرقم صفر له خصوصية خاصة ولذلك يجب اختبار البرنامج متضمناً الرقم صفر.

رقم الاختبار	البيانات	النتيجة المتوقعة
1	12،8	96
2	7،8-	56-
3	12،20	240-
4	7-،0	0
5	7،0-	0
6	0،8	0
7	8،0	0
8	0،0	0

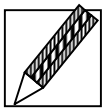
وهذا ما يكمل اختبار البرنامج طبقًا لمبدأ اختبار الصندوق الأسود.

لقد أوضحنا سبب حاجتنا إلى ثمانية مجموعات من بيانات الاختبار، وهذه المجموعات الثمانية معا عبارة عن المخرجات المتوقعة من الاختبار وتكون مواصفات الاختبار، هذا الاتجاه لابتكار بيانات اختبار الصندوق الأسود هو استخدام تكافؤ التقسيم؛ بمعنى النظر إلى طبيعة بيانات المدخلات لتحديد خواص مشتركة مثل تلك الخواص يسمى تقسيم أو تجزئة.

باختصار، إن قواعد اختيار بيانات اختبار الصندوق الأسود باستخدام تكافؤ التقسيم هي :

1) تقسيم كل قيم بيانات المدخلات.
2) اختيار بيانات تمثل الكل من كل جزء أو قسم (بيانات متكافئة) .
3) اختيار بيانات متاخمة (بالقرب من) الأقسام.

تدريب (2)



عزيزي الدارس، إذا كان لديك دالة مدخلاتها ثلاثة أرقام صحيحة (integers) ومخرجها الرقم الصحيح الأكبر، ارسم جدول الصندوق الأسود لاحتمالات المدخلات لهذه الدالة.

8. اختبار الصندوق الأبيض (التركيبي)

(The Box (Structural) Testing)

يستغل اختبار الصندوق الأبيض معرفة كيف يعمل الإجراء (Procedure) حيث يستخدم قوائم شيفرة البرنامج (سطور البرنامج المكتوب)، ومعرفة كيف يعمل الإجراء أي بنيته (تركيبه) تستخدم كقاعدة لابتكار بيانات الاختبار وبصورة مثالية يكتب المختبر النتائج المتوقعة للاختبار ومواصفات الاختبار ثم يجري تشغيل الإجراء (Execute it) وإدخال البيانات ثم مقارنة النتائج (المخرجات) مع النتائج المتوقعة.

لو أننا حاولنا الاختبار بتنفيذ كل ممر خلال الإجراء سوف نواجه سريعًا نفس المشكلة التي واجهتها في محاولة اختبار البرنامج اختبارًا شاقًا (مُجهدًا) عن طريق اختبار الصندوق الأسود، لكن هناك العديد من الحلول ،و حيث إن الإجراء يحتوي على حلقات فإن عدد الممرات المحتملة يتفجر (يزداد). لكن الحل البديل لخفض

عدد حالات الاختبار هو استخدام اختبار العبارة لاختبار إمكانية تصـــويت شـــخص من عدمه كمثال والتي تأخذ الشكل :

(سن تصويت) الجمهور.

لو السن 18 عد إلى (« يمكنه التصويت ») .

وإلاّ عد إلى (« لا يمكنه التصويت » .

إن مبدأ اختبار عبارة الصندوق الأبيض هو أن كل عبارة في البرنامج يجـــب أن تنفذ في وقت ما أثناء الاختبار ، في هذا الإجراء عبارة « لو » يتم تنفيذها دائمًا أيًا كانت البيانات لكن هناك قيمتان مختلفتان للبيانات نحن في حاجة إليهمـــا لتنفيـــذ النتيجتين المحتملتين للمقارنة وبالتالي تكون بيانات الاختبار المناسبة هي :

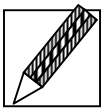
رقم الاختبار	البيانات	المخرجات/ النتائج
1	12	لا يمكنه التصويت
2	21	يمكنه التصويت

لاحظ أن اختبار الصندوق الأسود احتاج نفس نوع البيانات بالضـــبط لاختبـــار هذا الإجراء لكن تلك ليست الحالة بوجه عام، في اختبار الصندوق الأبـــيض نفـــس الجدال يُثار كما في اختبار الصندوق الأسود وهناك حالات خاصـــة تحتـــاج إلـــى فحص ومعاينة خاصة وبالتالي فإن اختبار الصندوق الأبيض يجب أيضًا أن يختـــار بيانات الاختبار التي :

- تكتشف القرارات التي تتحكم في اختيار الممرات .
- تهتم بأحداث الاختبار التي تؤخذ في حالات خاصة.

وكما هو الحال في اختبار الصندوق الأسود فهذا يسمى اختبار القيم المتاخمة (الحدّية) لاحظ أن اختبار العبارة سيجد الخطأ في إجراء المنتج المقدم مســـبقًا فـــي مثال عند مناقشة المشاكل العامة للاختبار.

تدريب (3)



الدالة التالية تعطي القيمة الكبرى لثلاثة أرقام صحيحة حدد جدول بيانات الصندوق الأبيض لهذه الدالة :

```
Int a,b,c
Int largest;
If (a>b)
      If(a>c)
            Largest =a;
Else
Largest c:
Else
If(b>c)
        Largest b;
          Else
Largest c:
```

9. اختبار بيتا (Beta Testing)

في اختبار بيتا يتم طرح أحد الإصدارات التمهيدية من البرنامج للعميل والذي يكون على دراية بأن المنتج به عيوب. ويُطلب من المستخدمين الإبلاغ عن العيبوب والاخطاء أو تسجيلها حتى يمكن تحسين المنتج قبل موعد طرحه في الأسواق ولقد اشتق الاسم « اختبار بيتا» من الحرف الثاني من الحروف اليونانية حيث يوصل الاسم فكرة أنه إجراء الاختبار الثاني والذي يتبع الاختبار داخل المنظمة التي قامت بالتطوير.

10. اختبار اندماج النظام

(System Integration Testing)

فيما سبق اعتبرنا أن التثبت من جودة الوحدة الواحدة أي إثبات جودة مكـون واحد فقط من مكونات البرنامج أو طريقة واحدة أو شيء مـا وهكـذا. وافترضـنا بصورة كلية أن هذا المكون عبارة عن جزء صغير جـدًا. وهـذه هـي الخطـوة (المرحلة) الأولى في التحقق من جودة أنظمة البرامج ، والتي بالطبع تتكـون مـن عشرات أو مئات المكونات الفردية. ومهمة اختبار الأنظمة الكاملة تسـمى اختبـار النظام أو اختبار الاندماج.

إن تطوير البرامج عن طريق تنفيذ التصميم الهندسـي يكـون بعـد وضـع المواصفات المفصلة لكل الأجزاء فالمشروع يُستأنف بواسطة التصـميم والشـيفرة المكونات الفردية ، ولكن المشكلة التي تظهر الآن هي « كيف يمكننا اختبـار هـذه المكونات وكيف يمكننا اختبار النظام؟».

للإجابة على هذا السؤال حاول التمعن في الاتجاهات الثلاثة التالية لاختبار النظام.

- الاختبار الكبير (Big Bang) : أي وضع كل المكونات معًـا بـدون اختبـار سابق ثم اختبار النظام بالكامل.
- **الاختبار الكبير المطور (Improved Big Bang)** : أي اختبار كل مكون مـن المكونات بصورة فردية ثم وضع كل المكونات معًا واختبار النظام بالكامل.
- **الاختبار المتزايد (Incremental)** : أي بناء النظام جزءًا جزءًا واختبار النظام الجزئي في كل مرحلة.

الاتجاه الأول أي الاختبار الكبير أو اختبار الأجزاء كوحدة واحدة يمثل كارثة حيث لا توجد طريقة سهلة لمعرفة أي الوحدات كانت سبب الخطأ ، وهناك أيضًـا مهمة شاقة لاكتشاف الأخطاء. أما الطريقة الثانية فتعتبر أفضل نسـبيًا لأنـه غيـر

وضع المكونات معًا فإننا يصبح لدينا الثقة فيه كمكونات فردية. والآن فإن ظهور أي خطأ من المحتمل أن يكون سببه التداخل والتفاعل بين المكونات وهناك بالتحديد تظهر مشكلة كبيرة في اكتشاف الأخطاء.

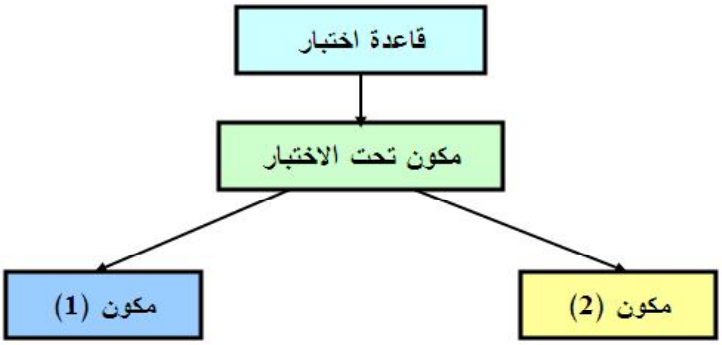
والبديل الأمثل هو استخدام أسلوب الاختبار المتزايد وفيه يتم أولاً اختبار مكون واحد من مكونات النظام ثم توصل وحدة ثانية مع الأولى ثم نختبر النظام ككل. وهكذا فإن إي خطأ يظهر فمن المحتمل أن يكون إما في الوحدة الجديدة أو في طريقة تداخل وتركيب الاثنين معًا. ثم نستمر في العمل بهذه الصورة بإضافة مكون واحد فقط في كل مره. وفي كل مرحلة أي خطأ سيظهر نفسه بسبب الوحدة الجديدة المضافة أو بسبب تركيبه مع النظام، وبهذه الطريقة فإن اكتشاف الأخطاء يصبح أسهل.

أن هناك طريقتين للاختبار المتزايد وهما الاختبار من الأسفل إلى الأعلى (التصاعدي) والاختبار من القمة إلى الأسفل (التنازلي) ، وفيما يلي مناقشة الطريقتين.

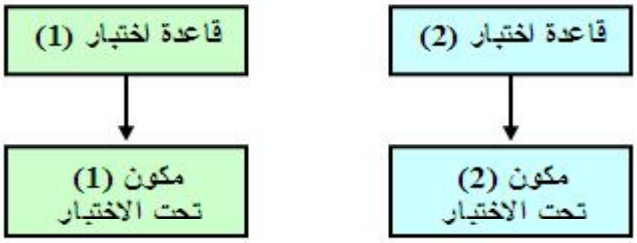
1.10 الاختبار من أسفل إلى أعلى

(Bottom-up Testing)

وهو أسلوب يبدأ بالمستوى الأدنى (الأسفل) للمكونات في النظام؛ وهي المكونات التي يستخدمها كل شيء آخر ولكنها في حد ذاتها لا تستخدم أي شيء، والمهمة الأولى هي إنشاء « حزام اختبار » أو « قاعدة اختبار » لكل مكون من المكونات، وهذا البرنامج المنشأ وظيفته الوحيدة هي تفعيل المكونات الخاضعة للاختبار (تحت الاختبار) بطريقة تتوافق مع دورها الحقيقي في النظام بالكامل، وبالتأكيد فإنه من المطلوب وجود بيانات اختبار خاصة. الشكل 5.5 يعرض مكونين في المستوى الأدنى أو الأسفل في النظام وأحزمة الاختبار الخاصة بهما.



شكل 5.5 : الاختبار من أسفل إلى أعلى كمرحلة وسطي



شكل 5.6 : الاختبار من أسفل إلى أعلى للمكونات الأدنى

ومما يجدر الإشارة إليه سريعًا هو أن أحزمة الاختبار في حد ذاتها قد تكون برنامجاً معقداً كبير الحجم ويتطلب تطويرًا ووقتًا للاختبار وكذلك فإن الإعداد لبيانات الاختبار قد تتطلب مجهودًا كبيرًا. وهاتان الملاحظتان تمثلان سقفًا كبيرًا للمشروع رغم أنهما لا يمثلان جزءً من النظام الكامل.

حينما يتم اختبار المكونات الأدنى من النظام بهذه الطريقة فإن الأجزاء تتحد لتكوين نظام فرعي يتم اختباره بطريقة مماثلة ، مرة أخرى باستخدام أحزمة الاختبار ، كما هو موضح بالشكل 5.6 ، ويستمر الإجراء حتى يتم اختبار النظام الكامل ككل في نهاية الأمر.

وطريقة الاختبار من أسفل إلى أعلى تعاني من العيوب الآتية :

①الوقت طويل نسبيًا في إنشاء أحزمة الاختبار وبيانات الاختبار ،والأسوأ من ذلك أن كلا منهما يتخلص منه عند إتمام عملية الاختبار، وهذا يتم تشبيهه بعمل النجار الذي يصنع لنفسه مجموعة من أطقم الأدوات والمعدات الخاصة لبناء منزل ما ثم يحطم هذه المعدات والأدوات بعد انتهاء بناء المنزل (المقصود من التشبيه هو المجهود الضائع). بدلاً على ذلك فإن قواعد الاختبار وبيانات الاختبار يتم الإبقاء عليها والاحتفاظ بها لاستخدامها عند الحاجة إلى إضافة التحسينات أو التعديلات على النظام للتأكد من أن المكونات التي كانت تعمل في السابق مازالت تعمل وفي هذه الحالة فإن مادة الاختبار يجب أن تخزن وتتم صيانتها دوريًا وهو ما يتطلب لجهدًا كبيرًا أيضًا.

② الأخطاء التي تكتشف في مراحل الاندماج لاختبار النظم الفرعية تتطلب التكرار للعملية بالكامل من تصميم ووضع شيفرة واختبار الوحدة. والأنظمة الفرعية أو المكونات قد يكون من الواجب إعادة عملها مرة أخرى كلما تم اندماج النظام ولا عجب في أن اختبار النظام يستغرق النسبة الأكبر (ما بين الثلث والنصف) للمعدل الزمني للمشروع. ولكن السؤال هنا لو أنَ كل الأجزاء الفردية تعمل بصورة صحيحة فلماذا لا تعمل جميعها عند اتحادها ؟. الإجابة هي بالطبع تحديدًا في التداخلات التي تكشفها الأخطاء وهناك موقف أكثر تعقيدًا حيث تعالج الأجزاء البيانات المشتركة أو تراكيب البيانات.

③ اختبار النظام بالكامل يتم بالقرب من نهاية المشروع ، لذلك فإنه غالب الأخطاء الكبيرة في تصميم النظام لا تكتشف حتى اكتمال الوقت المحدد، وهذه الاكتشاف قد يؤدي إلى إعادة بناء الأجزاء الكبيرة للنظام في وقت حرج جدًا .

④ لا يوجد نظام عمل مرئي حتى اكتمال المرحلة الأخيرة وهي اختبار النظام،

ومن الحقيقي أن هناك مكونات ونظم فرعية تم اختيارها ولكن لا يوجد شيء يمكن إيضاحه وعرضه للعميل باعتباره رؤية محددة لما قد يقوم به النظام فعليًا.

كل هذه المشاكل يمكن أن تزيد المشكلات التي يكافح فريق العمل لوضع نهاية لها، ولا عجب أن الساعات الطويلة من العمل المتواصل تؤدي إلى الأخطاء التي تظهر كما لو أن الظروف تتآمر ضد المشروع ، لكن الاختبار من القمة للأسفل يساعد على تفادي بعض هذه المشكلات كما سنشرح الآن.

2.10 الاختبار من أعلى لأسفل

(Top-down Development)

تُعد طريقة قيمة لتطوير البرمجيات حيث تعمل وفقا للقواعد التالية :

① يبدأ التصميم للمكونات العليا للبرنامج أو النظام ، والتي تمثل في الحقيقة واجهة الاستخدام للنظام.

② يتم تشفير (تكويد) المكونات العليا للنظام.

③ يتم توليد بيانات الاختبار للمكونات العليا.

④ يستمر التنفيذ عن طريق اختبار مكونات ذات مستوى أدنى (كانت في البداية بقايا جذرية) لتكوين وتشفير وتجسيد النظام . بصورة عامة فإنه في أي مرحلة من مراحل التطور يوجد :

- مكونات الجزئية التي تحت الاختبار.
- البقايا الجذرية.

وعملية التطور هذه لن تستمر لأن هناك صعوبات قد تواجهها مثل :

أولاً : لن يتم التطور على أساس صلب (مستوى ثم مستوى) فإن بعض المكونات ذات المستوى الأدنى تحتاج إلى تصميم ثم تشفير واختبار في مراحل متقدمة ،

ومثال لذلك تطوير أجزاء البرنامج الذي يتفاعل مع المستخدم، حيث يظهر تصميم وشكل الشاشة للمستخدم الأخير للبرنامج الذي ربما يطلب التعديل.

ثانياً : يفضل بعض المبرمجين طرقاً مختلفة في التعامل فهم يخافون من ترجمة بعض التصاميم إلى رموز يتم اختبارها حيث يجدون نتيجة لتطوير مكونات ذات مستوى أدنى فإن هناك خطأ ما في المكونات ذات المستوى الأعلى ، وهذا يتطلب إعادة التصميم وإعادة التشفير والاختبار، والطريقة البديلة التي تبقى العديد من الفوائد هي إكمال التصميم لكل نظام أو برنامج قبل البدء في التشفير والاختبار من أعلى لأسفل.

أخيراً :هناك رأي آخر معارض للحد من استخدم هذه الوسيلة فهو من السهل للمبرمج أن يختبر المكونات كلاً على حدة بدلاً منها باعتبارها جزءً من النظام ذى المستوى الأعلى. وبالطبع في بعض الأحيان يكون من الصعب أن نختبر من أعلى لأسفل لعدم توفر معلومات الاختبار ، وفي مثل هذه الحالة يفضل تجسيد المكون الجديد في وقت مناسب لاختبار حدودها البينية في المكان الجديد.

هناك بعض الملاحظات في التنفيذ من أعلى لأسفل وهي :

① الاكتشاف المبكر للعيوب الكبيرة : حيث إنه من أهم المشاكل التي تواجه التطوير بأكمله مما يتطلب إعادة تشكيل الأجزاء الكبيرة من النظام، بجانب أن التطوير من أعلى لأسفل يضمن أن المكونات الكبيرة يتم اختبرها أولاً حيث يمكن تصحيح الأخطاء دون إعادة الكتابة،و بذا فإن هذه الطريقة تساعد في تقليل الجهد المبذول وإنجاز العمل في وقته، ويعتقد البض أن الإجراءات ذات المستوى الأعلى من البرنامج هي المهمة بينما يظن البض أن ذات المستوى الأدنى يسهل تجاهلها رغم أنها الأهم.

② المصداقية والاعتمادية : عندالاختبار من أعلى لأسفل فإن مكونات برنامج الحاسب الآلي يتم اختبارها عدة مرات مما يسهل اختبار الأجزاء ذات المستوى الأدنى ويسهل اكتشاف الأخطاء.

③ إزالة الأخطاء : بهذه الطريقة يسهل معرفة موقع الأخطاء، ولأن الجذور يتم استبدالها كل واحدة على حده بمكون جديد فإن المشكلة قد تكون في أحد هذه المكونات الجديدة أو في سطحها المشترك مع آخرى ذات مستوى أعلى وعلى العكس في حالة من أسفل لأعلى فإن العيوب تكون في المكون الجديد أو في الأسطح الجديدة (وهذا النوع من الاختبار يسمى الاختبار التزايدي).

④ الجانب المعنوي : في هذه الطريقة يكون العمل محسوساً وفي مرحلة مبكرة يتم التطوير مما يطمئن المديرون والعملاء.

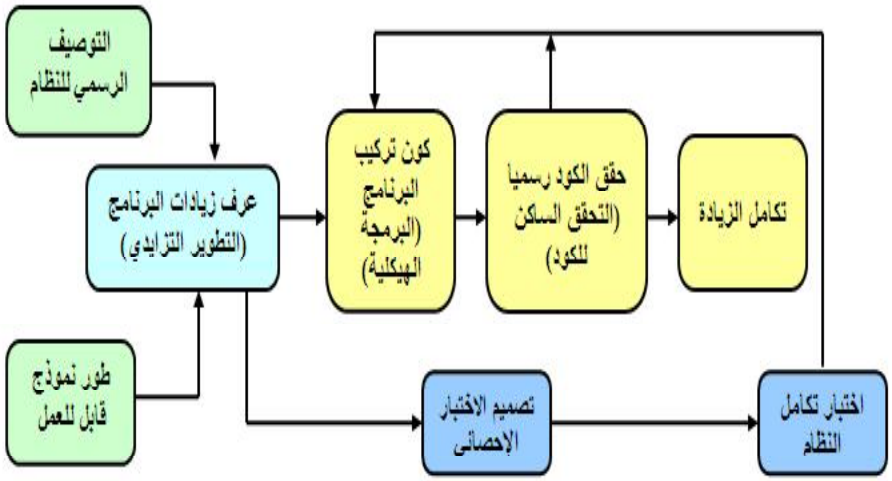
⑤ الطباعة المبكرة : في هذا النوع يمكن اعتبار نص التنفيذ من أعلى لأسفل الأولى للبرنامج كطباعة أولية للمستخدم بحيث يستطيع المستخدم التعليق على البرنامج ويقترح التغييرات قبل العمل.

⑥ زمن البرنامج : يكون في هذا النوع من أعلى لأسفل الزمن المستغرق أقل نسبة لسببين ورد ذكرهما ، إضافة إلى بناء أساس اختباري ومعلومات اختبارية . كما أن الأساس الاختباري لا يحتاج إلى تشييد حيث يكون قد تم تشييده عن طريق الجزء ذي المستوى العلوي في البرنامج الذي تم اختباره أما البقايا الجذرية فتحتاج لذلك وتكون بسهولة تتناسب مع هذه الوسيلة خاصة المشاريع التي يعمل فيها عدة مبرمجين . يبدأ تطوير هذه الطريقة بالتنفيذ الكامل للمستويات العليا للبرنامج وهذا عادة يتم بواسطة مهندس برمجة واحد ، أما بقية المبرمجين في هذه المرحلة يمكن أن يخصص لكل واحد منهم تطوير المستويات السفلى المتوقعة مع تحديد العديد من المشاكل التي تحدث في التطوير من أسفل لأعلى. لذا يفضل تأخير هذا التوزيع حتى إكمال إجراءات المستويات العليا للبرنامج.

11. تطوير البرمجيات بطريقة الغرفة النظيفة

(Clean Room Software Development)

تطوير البرمجيات بطريقة الغرفة النظيفة (ميلز وآخرون 1987م ، لينجر 1994م ، بووال وآخرون 1999 م) هي فلسفة تطوير برمجيات مبنية على تفادي الأخطاء في البرمجية باتباع عملية فحص صارمة ، الهدف منها تطوير برمجية خالية من الأخطاء. إن تسمية " الغرفة النظيفة" مشتقة من التشبيه بوحدات تصنيع أشباه الموصلات حيث يتم في هذه الوحدات (الغرفة النظيفة) تجنب وجود أخطاء بالتصنيع في بيئة فائقة النظافة ، وتعتمد فلسفتها على تفادي الخلل أثناء التطوير بدلا من استبعاده ، وسنقوم بمناقشة هذا الموضوع في الوحدة لأن هذه الطريقة استبدلت اختبار الوحدة لأجزاء النظام بواسطة الفحوصات لمعاينة انسجام هذه الأجزاء مع مواصفاتها، ويظهر في الشكل 5.7 نموذج لعملية تطوير البرامج بأساليب الغرف النظيفة المشتقة من الوصف الذي قام به الباحث لينجر (1994م).



شكل 5.7 : عملية تطوير البرامج بأساليب الغرف النظيفة

- **خصائص عمليات الغرف النظيفة.**

(Clean-room Process Characteristics) :

إن طريقة الغرفة النظيفة مصممة لدعم الفحص الصارم للبرنامج ، حيث يتم إنتاج نموذج نظام مبني على الحالة يستخدم كمواصلة للنظام وتتم تنقيته بواسطة عدد من النماذج ليصبح برنامجاً قابلاً للتشغيل ، كما أن المنهج المستخدم في التطوير يعتمد تحولات مُعرَفة بشكل جيد يحاول فيها الاحتفاظ بالوضع الصحيح عند كل تحول إلى التمثيل الأكثر عموماً وتدعم فحوصات البرنامج بعدد من المعادلات الحسابية الصارمة توضح أن النتائج من البرنامج المعدل منسجم في مدخلاته.

إن المعدلات الحسابية المستخدمة في عمليات الغرف النظيفة أقل متانة من البراهين الرياضية المنهجية ، ذلك أن البراهين الرياضية المنهجية مرتفعة التكاليف في تطويرها لأنها تعتمد على معرفة القواعد المنهجية للغة البرمجة لكي يتم تطوير نظريات تطابق البرنامج ومواصفته المنهجية ، ومن ثم يجب أن تبرهن هذه النظريات رياضياً باستخدام برامج كبيرة ومعقدة ، وبسبب تكلفتها العالية والمهارات المتخصصة المطلوبة ، فإن البراهين تطور دائماً للتطبيقات الحساسة الخاصة بالسلامة أو الحساسة أمنياً.

وبصفة عامة يعتمد منهج تطوير البرمجيات عن طريق الغرف النظيفة على خمسة خصائص هامة يمكن تلخيصها كمايلي :

① التوصيف الرسمي للبرمجية **(Formal Specification)** : توصف البرمجية التي يتم تطويرها بصفة رسمية (منهجية) ويستخدم نموذج انتقال الحالة الذي يظهر الاستجابة للمؤثرات للتعبير عن المواصفة.

② البرمجة الهيكلية **(Structure Programming)** : يستخدم فقط عدد محدد من مركبات التحكم والبيانات التجريدية حيث إن عملية تطوير البرنامج هي عملية تنقية تصاعدية للمواصفة ويتم استخدام عدد محدد من المركبات ، الهدف منها تطبيق تحولات تحفظ سلامة المواصفة ، وبالتالي لترميز البرنامج.

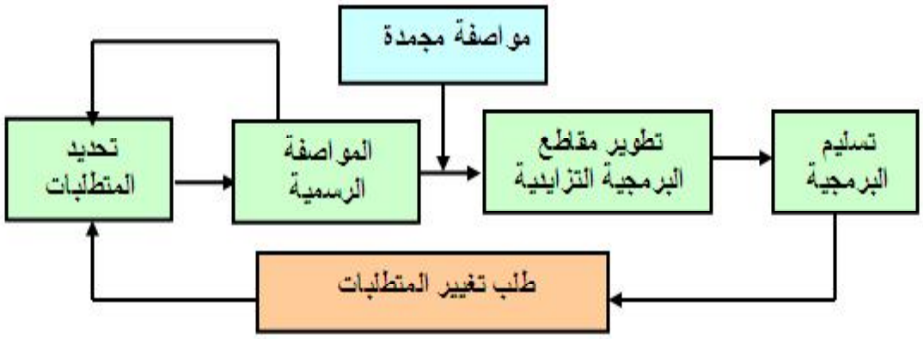
③ التحقق الساكن **(Static Verification)** : يتم التحقق الساكن للبرمجة المطورة باتباع فحوصات صارمة للبرمجية ، ولا يوجد هنا اختبارات للوحدة أو الموديلات لمكونات الشيفرة.

④ الاختبار الإحصائي للنظام **(Statistical Testing)** : يتم اختبار البرمجية الإضافية المدمجة إحصائيا ، لتحديد اعتماديته ، وهذه التحاليل الإحصائية مبنية على منظور تشغيلي يتم تطويره بالتوازي مع مواصفة النظام.

⑤ التطوير التزايدي **(Incremental Development)** : وفيها يتم تجزي البرمجية التي يتم تطويرها إلى جزئيات يحقق كل جزء منها على حدة ويصادق عليه باستخدام طريقة الغرفة النظيفة ويتم عمل هذه التقسيمات مع تضمين مدخلات العميل في وقت مبكر من العملية.

إن التطوير التزايدي ، الموضح في الشكل 5.8 ، يختص بإنتاج وتسليم برمجية علي مقاطع تزايدية منفصلة ، ويمكن أن يتم تشغيل المقطع بأوامر يقوم بإصدارها المستخدم ويكون المقطع نظاماً قائماً بذاته ومحدداً ، ويقوم المستخدمون بتقديم آرائهم عن النظام وباقتراح التغييرات المطلوبة. إن تطوير المقاطع التزايدية مهم جداً لأنها تقلل من توقف التشغيل لعملية التطوير الناجمة عن التغييرات التي يطلبها المستخدمون للمتطلبات ، فعندما تتم المواصفة في وحدة واحدة توقف التغييرات الخاصة بالمتطلبات التي يطلبها المستخدمون (وهو أمر لابد منه) للمواصفة وعملية التطوير بكاملها ، وبالمقابل ، فمع التطوير المقطعي فإن

المواصفة للمقطع تكون مجمدة بينما يتم قبول طلبات التغير الخاصة بباقي النظام ويتم تسليم مقطع البرمجية عند الاستكمال.



شكل 5.8 : شكل توضيح لعملية التطوير التزايدي

إن منهج التطوير المقطعي في عملية الغرفة النظيفة هو لتسليم الأجزاء الحساسة في مقاطع يتم تسليمها في وقت مبكر، وتضاف الوظائف الأقل حساسية الى المقاطع اللاحقة ، وبذلك تتوفر للعميل فرصة تجربة هذه المقاطع الحساسة قبل أن يتم تسليم النظام الكامل ، فإذا اكتشف وجود مشاكل في المتطلبات فإن العميل يقوم بإخطار فريق التطويربذلك ويطلب إصداراً من المقطع ، وهذا معناه أن الوظائف الأكثرأهمية بالنسبة للعميل تلقي المراجعة والتصحيح الأكبر ، كما أن المقاطع الأخرى تضاف إلى المقاطع الموجودة سابقاً متى ما يتم الانتهاء منها ومن ثم تتم تجربة النظام المتكامل ، ولذلك فإن المقاطع الموجودة سابقاً تجرب مرة أخرى بأوضاع اختبارات جديدة كلما تمت إضافة آخر.

- **فرق عمل الغرفة النظيفة (Cleanroom Process Teams) :**

هناك ثلاثة فرق عندما يتم القيام بتطوير أنظمة غرف نظيفة كبيرة الحجم وهي :

① فريق المواصفات : مسئول عن تطوير وصيانة مواصفة النظام ، هذا الفريق يقوم بإنتاج المواصفات المتعلقة بالعملاء (تحديد المتطلبات)

والمواصفات الحسابية للتحقق وفي بعض الحالات وعند الانتهاء من المواصفات فإن الفريق يشارك في عملية التطوير.

② فريق التطوير : هذا الفريق مسئول عن التطوير والتحقق للبرمجية ، ولا يتم تشغيل البرمجية خلال مرحلة التطوير ويستخدم أسلوب منهجي هيكلي للتحقق مبني على فحص الكود معزز بمحاولات تصحيحية.

③ فريق التحقق والمصادقة : هذا الفريق مسئول عن تطوير مجموعة من الاختبارات الإحصائية لاختبار البرمجية بعد تطويرها ويتم بالتوازي معها تطوير حالات الاختبار وتستخدم حالات الاختبار للشهادة على اعتمادية البرمجية ويمكن أن يستخدم نموذج الاعتمادية.

- ميزات أساسية للبرمجيات المنتجة بالغرف النظيفة :

إن استخدام طريقة الغرف النظيفة أدت إلى تطوير برمجيات قليلة الأخطاء وليست أكثر تكلفة من الأنواع التقليدية ، وقد ناقش الباحثان كوب وميلز (1990م) عدداً من المشاريع التي استخدمت فيها هذه الطريقة وكان فيها عدد قليل من الأخطاء في الأنظمة التي تم توريدها ، وقد كانت تكاليف هذه المشاريع مماثلة مع المشاريع الأخرى التي استخدمت فيها الطرق التقليدية.

إن التحقق الساكن فعال في التكلفة باستخدام أسلوب الغرف النظيفة ، وذلك لأن النسبة الكبيرة من الأخطاء يتم اكتشافها قبل التنفيذ ولا تدخل في البرمجية المطورة ، وقد وجد الباحث ينجر (1994م) أن نسبة الأخطاء المكتشفة في المتوسط ، كانت فقط 3.2 خطأ لكل ألف سطر من نص شيفرة المصدر خلال اختبار المشاريع المطورة باستخدام أسلوب الغرفة النظيفة ، كما أن التكاليف الإجمالية للتطوير لم ترتفع وكان الجهد المطلوب في الاختبار وتصحيح الأخطاء أقل في هذه الحالات.

كما قام الباحثون سلبي وآخرون (1987م) بمقارنة التطـوير (بالاسـتعانة بطلبة لمطوري البرنامج) باستخدام أساليب الغرف النظيفة والطرق التقليدية ، حيث وجد أن البرامج التي تم تطويرها كانت في مجملها أكثر جودة من تلـك التـي تـم تطويرها باستخدام الأساليب التقليدية ــ شيفرة المصدر كانت تحتوي على تعليقـات وقد كانت أبسط في التركيب إضافة إلى الالتزام بالمواعيد المحددة.

إن التطوير باستخدام أساليب الغرف النظيفة يعمل جيـدا عنـدما يقـوم بـه مهندسون متخصصون وملتزمون ، وهذا مقتصر فقط علـى المنشـآت المتقدمـة تقنياً. إن التقارير عن النجاحات المحققة بـطريقة الغرف النظيفة فـي الصـناعة ، كانت في أغلبها وليست كليا من المطورين ، لذلك فإننا لا نعلم ما إذا كان بالإمكـان تطبيق العملية لمنشآت تطوير البرمجيات الأخرى علماً بأن هذه المنشـآت لـديها دائماً عدد أقل من المهندسين ،و لذلك فإن تطبيق أساليب الغرف النظيفة فـي هـذه المؤسسات لا يزال حتى الآن تحدياً.

الخلاصة

اشتملت هذه الوحدة على:

– تعريف كل من : التحقق: وهويعني إنتاج برمجية خالية من الأخطاء.

المصادقة:وهي تعني التأكد من تلبية البرنامج لمتطلبات العملاء.

– طبيعة الأخطاء: تعرفنا هنا على أهم أنواع الأخطاء التي قد تظهر عند تشفير (تكويد) أحد المكونات:وهي عدم إعطاء قيم بدائية للبيانات ،و تكرار الحلقات عدد مرات غير صحيح ،و أخطاء في القيم الحدية.

– التحقق والمصادقة الساكنة والديناميكية: وتشمل طريقتين للفحص والتحليل:

* فحص البرنامج:هذا ويشمل التحليل والتدقيق على مخططات البرنامج المختلفة مثل مستندات المتطلبات ،و مخطط التصميم ، وشيفرة البرنامج المصدر.

* اختبار البرنامج:هذا ويشمل تنفيذ تطبيق البرنامج عن طريق بيانات اختيارية ومعاينة مخرجات البرنامج ، وهناك اختبارات متعددة منها : اختبار الخلل ، والاختبار الإحصائي.

– اكتشاف تصحيح الأخطاء:وهو العملية التي تحدد مكان الأخطاء حيث يتم تصحيحيها.

– التخطيط للتحقق والمصادقة:وقد تعرفنا هنا على عدد من النماذج:

* نموذج الاختبار والتطوير: حيث يتم تقسيم أنشطة التحقق والمصادقة إلى عدد من المراحل بحيث تقاد كل مرحلة من الاختبارات التي حددت لها اكتشاف مطابقة البرنامج لمواصفاته وتصميمه.

* هيكل خطة اختبار البرمجيات: تأخذ هذه الخطة في الاعتبار كميات كبيرة من الإعمال المضاعفة (حالات الطوارئ) بحيث يتم الإحاطة الكاملة بالتأخيرات الخاصة على التصاميم التنفيذ.

– فحوصات البرامج: مراجعات هدفها اكتشاف الأخطاء التي قد تكون موجودة في البرنامج ، وتعرفنا هنا على:

* فرق التفتيش (الفحص): يقوم بها في العادة فريق مكون من أربعة أشخاص.

* الشروط المسبقة للتفتيش:وهي أن يكون هناك مواصفات دقيقة ،و أن يكون أعضاء فريق الفحص على دراية بمعايير المؤسسة و أن يكون هناك نسخة صحيحة من حيث القواعد وحديثة من حيث البرنامج الذي يراد فحصه ،كما يجب أن تتوقع الإدارة أن عملية الفحص ستزيد من التكلفة،و يجب أن لا تستخدم الإدارة عملية الفحص لتقويم العاملين.

* عملية التفتيش (الفحص).

* قوائم الفحص.

* معدلات التفتيش.

– التحليل الآلي الساكن:وهو أدوات برمجية تمر على جميع محتويات نص البرنامج المصدر وتكشف الأخطاء التي فيه . وتعرفنا هنا أيضاً على:

* فحوصات التحليل الساكن: ويقصد بها جذب انتباه الشخص القائم على التحقيق للكائنات الشاذة في البرنامج مثل استخدام متغيرات لم يتم استهلالها.

* مراحل التحليل الساكن: ويشمل : تحليل مسار التحكم ، تحليل استخدام البيانات ، تحليل الواجهة ، تحليل تدفق المعلومات ، تحليل المسار.

* استخدامات التحليل الساكن.

– اختبار الصندوق الأسود:وقد تعرفنا هنا على قواعد اختيار بيانات اختبار الصندوق الأسود باستخدام تكافؤ التقسيم و هي:

* تقسيم كل قيم بيانات المدخلات * اختيار بيانات تمثل الكل من كل جزء أو قسم (بيانات متكافئة) * اختيار بيانات متاخمة (بالقرب من) الأقسام.

اختبار الصندوق الأبيض (التركيبي): يستغل اختبار الصندوق الأبيض معرفة كيف يعمل الإجراء (procedure) حيث يستخدم قوائم شفرة البرنامج، اختبار الصندوق الأبيض يجب أيضاً أن يختار بيانات الاختبار التي:

* تكتشف القرارات التي تتحكم في اختيار الممرات.

* تهتم بأحداث الاختبار التي تؤخذ في حالات خاصة.

– اختبار بيتا: وفيهيتم طرح احد الإصدارات التمهيدية من البرنامج للعميل والذي يكون على دراية بأن المنتج به عيوب.

– اختبار اندماج النظام: توجد ثلاثة اتجاهات لاختبار النظام:

* الاختبار الكبير:وهو وضع كل المكونات معاً بدون اختبار سابق ثم اختبار النظام بالكامل.

* الاختبار الكبير المطور: وهواختبار كل مكون من المكونات بصورة فردية ثم وضع كل المكونات معاً واختبار النظام بالكامل.

* الاختبار المتزايد:وهو بناء النظام جزءاً جزءاً واختبار النظام الجزئي في كل مرحلة،غير أن هناك طريقتين للاختبار المتزايد: الاختبار من الأسفل إلى الأعلى (التصاعدي)، والاختبار من القمة إلى الأسفل (التنازلي).

– تطوير البرمجيات بطريقة الغرفة النظيفة: وهوفلسفة تطوير برمجيات مبنية على تفادي الأخطاء في البرمجة باتباع عملية فحص صارمة ، وتتلخص خصائص الغرف النظيفة في:

التوصيف الرسمي للبرمجة ،و البرمجة الهيكلية ،و الاختبار الإحصائي للنظام ، والتطوير التزايدي.

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس،،

بعد أن فرغت من دراسة الوحدة الخامسة ينتقل بك المقرر إلي الوحدة السادسة وعنوانها "" **مبادئ تقدير تكلفة البرمجيات**" ، وفيها يتم توضيح أهمية التقدير الدقيق لتكلفة المشاريع البرمجية ، والعوامل التي تؤثر على دقة هذا التقدير ، مع شرح وافٍ لأهم الطرق والمنهجيات المختلفة التي تستخدم لتقدير الجهد والتكاليف وفترة الجدولة الزمنية ، والمتغيرات الأساسية التي تدخل في ذلك ، مع ذكر بعض الإرشادات والأفكار المفيدة لتقدير تكلفة المشروعات البرمجية.

إجابات التدريبات

تدريب (1)

هو أن الهدف الأساسي للفحوصات هو اكتشاف الأخطاء التي قد تكون موجودة في البرنامج بدلاً من إعطاء اعتبارات لأمور تتعلق بالتصميم الكلي للمشروع ، وقد تكون الأخطاء أما أخطاء منطقية أو نتيجة لعدم المطابقة للمعايير

المحددة للمنشأة أو للمشروع ، وبالعكس من ذلك فقد تكون الأنواع الأخرى من المراجعات ذات علاقة بالجداول أو التكاليف أو تقدم العمل مقابل مراحل معينة ومحددة مسبقة أو بتقويم ما إذا كان البرنامج يفي بأهداف المنشأة المحددة له.

تدريب (2)

لم يشترط التخصيص أي حدود في حجم الأرقام ، لذا نقترح أقصى مدى ممكن لقيم الأرقام الثلاثة (ابتداء من السالب وحتى الموجب ومروراً بالقيمة صفر) ، وقد اختارنا قيما لكل عدد بحيث يتم تمثيل هذا المدى (قيمة سالبة والقيمة صفر وقيمة موجبة) ، كما هو موضح أدناه

الرقم الأول	**9-**	**صفر**	**+34**
الرقم الثاني	**56-**	**صفر**	**+4**
الرقم الثالث	**2-**	**صفر**	**+123**

لذا فإن بيانات الصندوق الأسود يعتبر المجموعات الممكن تكوينها من هذه القيم:

رقم الاختبار	**بيانات الاختبار (مجموعات قيم الاختبار)**			
1	**9-**	**56-**	**2-**	**المجموعة**

				الأولى
2	9-	56-	صفر	المجموعة الثانية
3	9-	56-	123	المجموعة الثالثة
4	9-	صفر	2-	المجموعة الرابعة
5	9-	صفر	صفر	المجموعة الخامسة
وهكذا حتى يتم الانتهاء من جميع المجموعات التي يمكن تكوينها من الأرقام المختارة في الجدول السابق				

في النهاية تأكدنا من أن التخصيص لم يشترط إذا ما كان اثنان منهما أو أكثر متساووين و لذا فإننا نتساءل ونشك في التخصيص مع العميل أو نجد افتراضاً معقولاً لذلك . فلنفترض أن اثنين أو أكثر من هذه القيم متساوية فإن البرنامج سيعرض القيمة الكبرى كما أن معلومات الاختبار هي :

رقم الاختبار	بيانات الاختبار		
1	4	4	2
2	2	5	5
3	6	4	6
4	3	3	3

تدريب (3)

يوجد أربعة مسارات خلال البرنامج والتي يمكن تمارس أو تنفذ ببيانات الاختبار الآتية :

رقم الاختبار	بيانات الاختبار		
1	3	2	1
2	3	2	5
3	2	3	1
4	2	3	5

مسرد المصطلحات

الوحدة الخامسة:

اختبار البرنامج Software Testing :

يشمل تنفيذ تطبيق للبرنامج عن طريق بيانات اختباريه ومعاينة مخرجات البرنامج وسلوكها التشغيلي للتأكد من أن البرنامج يفي بما هو مطلوب منه.

الاختبار الإحصائي Statistical Testing :

ويستخدم لاختبار أداء البرنامج والاعتماد عليه ولكشف طريقة عمله تحت الظروف التشغيلية المختلفة.

فحوصات البرامج Program Inspections:

فحوصات البرامج هي مراجعات هدفها اكتشاف الأخطاء والتي قد تكون موجودة في البرنامج.

تطوير البرمجيات بطريقة الغرفة النظيفة:

Clean Room Software Development :

هي فلسفة تطوير برمجيات مبنية على تفادي الأخطاء في البرمجية بإتباع عمليـــة فحص صارمة ، الهدف منها تطوير برمجية خالية من الأخطاء.

المصطلح بالإنجليزية	معناه بالعربية
Automated Static Analysis	التحليل الآلي الساكن
Beta Testing	اختبار بيتا
Big Bang Test	الاختبار الكبير
Black Box (Functional)Testing	اختبار الصندوق الأسود
Bottom-up Testing	الاختبار من أسفل إلى أعلى
Clean Room Process Teams	فرق عمل الغرفة النظيفة
Clean Room Software Development	تطوير البرمجيات بطريقة الغرفة النظيفة
Constrains	القيود
Control Faults	أخطاء تحكم

المصطلح بالإنجليزية	معناه بالعربية
Control Flow Analysis	تحليل مسار التحكم
Data Base Analysis	تحليل استخدام البيانات
Data Faults	أخطاء بيانات
Defect Testing	اختبار الخلل
Fault Class	فئة الخطأ
Formal Specification	التوصيف الرسمي للبرمجية
Hardware and Software Requirements	متطلبات الأجهزة والبرمجيات
Improved Big Bang Test	الاختبار الكبير المطور
Incremental Development	التطوير التزايدي
Incremental Test	الاختبار المتزايد
Information Flow Analysis	تحليل تدفق المعلومات
Input/Output Faults	أخطاء دخل/ خرج
Inspection Check	الفحص المطلوب
Inspection Checklists	قوائم الفحص
Inspection Pre-condition	الشروط المسبقة للتفتيش
Inspection Rates	معدلات التفتيش
Inspection Teams	فرق التفتيش(الفحص)
Interface Analysis	تحليل الواجهة
Interface Faults	خطأ واجهة
Marketing Environments	البيئة التسويقية
Online-Systems	أنظمة الوقت المباشر
Path Analysis	تحليل المسار
Program Inspections	فحوصات البرامج
Programs Analyzers	محللات البرامج
Requirements Traceability	تعقب المتطلبات

المصطلح بالإنجليزية	معناه بالعربية
Software Function	وظيفة البرنامج
Software Inspection	فحص البرنامج
Software Testing	اختبار البرنامج
Software Verification and Validation	التحقق والمصادقة على صحة البرمجيات
Static Analysis Checks	فحوصات التحليل الساكن
Static and Dynamic Verification and Validation	التحقق والمصادقة الساكنة والديناميكية
Static Verification	التحقق الساكن
Statistical Testing	الاختبار الإحصائي
Statistical Testing	الاختبار الإحصائي للنظام
Storage Management Faults	أخطاء إدارة الذاكرة
Structure of a Software Test Plan	هيكل خطة اختبار البرمجيات
Structure Programming	البرمجة الهيكلية
System (Integration) Testing	اختبار (اندماج) النظام
Test Recording Procedures	إجراءات تدوين الاختبارات
Testing and Debugging	اكتشاف وتصحيح الأخطاء
Testing Schedule	جدولة الاختبارات
The Inspection Process	عملية التفتيش
The Nature of Errors	طبيعة الأخطاء
Top-down Development	الاختبار من أعلى لأسفل
User Expectations	توقعات المستخدمين
Verification and Validation Planning	التخطيط للتحقق والتصديق
V-Model Development	نموذج الاختبار والتطوير V
White Box (Structural) Testing	اختبار الصندوق الأبيض (التركيبي)

المراجع

أولاً : المراجع العربية

[1] مهندس عبد الحميد بسيوني ، "أساسيات هندسة البرمجيات" ، دار الكتب العلمية للنشر والتوزيع ، القاهرة ، 2005 م.

ثانياً : المراجع الإنجليزية :

[2] Ian Somerville , "Software Engineering", Addison Wesley, 2001.

[3] Ronald J. Leach,, "Introduction to Software Engineering", CRC Press, 1999.

[4] Douglas Bell , "Software Engineering A Programming Approach", 3rd Edition, Addison Wesley.

[5] **Shari Pfleeger, "Software Engineering - Theory and Practice", 2nd Edition.**

ثالثاً : مواقع على شبكة الإنترنت تم الاستفادة منها :

[6] www.icis.hq.dla.mil/systemdoc/FD/plan/chapter2.htm ,
" Chapter 2 - Software Verification and Validation ".

[7] http://csdl.computer.org/comp/mags/so/1989/03/s3010abs.htm ,
" Software Verification and Validation: An Overview"

[8] http://www.fabricadesoftware.cl/documentos/ESA/PSS0510.pdf
" Guide to software verification and validation"

[9] WWW. sunset.usc.edu/~neno/cs477_2003/April10.ppt
"Verification and Validation"

[10] www.comp.lancs.ac.uk/computing /
resources/IanS/SE7/Presentations/PPT/ch22.ppt
"Verification and Validation"

[11] www.cse.mrt.ac.lk/lecnotes/cs302/ch19-v-v.ppt
"Verification and Validation"

[12] www.i-u.de/schools/heinz/ EVOOC/2002-3-IT260/Slides/Session-10.ppt
"Verification and Validation"

[13] www.int.gu.edu.au/courses7014 /int/lectures/QualityPlan2.ppt
" Software Quality Planning "

الوحدة السادسة

مبادئ تقدير تكلفة البرمجيات

محتويات الوحدة

المقدمة

تمهيد

عزيزي الدارس ،،

أهلاً بك في الوحدة السادسة من مقرر " هندسة البرمجيات(1) " ، مبـــادئ تقـــدير تكلفة البرمجيات:

وتحوي هذه الوحدة على الموضوعات الأساسية التالية:

– أهمية التقدير الدقيق لتكلفة المشاريع البرمجية هـــو ويتضـــمن تقـــدير المعـــالم الأساسية وهي الحجم والموارد وفريق التطوير والجداول الزمنية والتكلفة.

– العوامل التي تؤثر على دقة تقدير تكلفة المشاريع البرمجية وهي درجـــة دقـــة تقدير حجم وافق المشروع ووقت عملية التقدير ودقة تحديد متطلبات المشـــروع ومقدار ثبات ظروف بيئة التطوير ودقة ترجمة هذا الحجم وهذه المتطلبـــات إلـــى جهد بشري وقدرات وكفاءات وإنتاجية فريق العمل ، حجم البرمجيـــات الجـــاهزة التي يمكن إعادة استخدامها في المشروع الجديد ومدى درايـــة وخبـــرة مـــديري المشاريع وطاقم التطوير بالتغير الهائل في التقنيات المستخدمة فـــي المشـــاريع . وسنتعرف في هذا القسم أيضاً على المسئول عن تقدير تكلفة المشروعات.

– طرق تقدير حجم البرمجية : بصفة عامة يوجد أربـــع طـــرق لتقـــدير حجـــم البرمجية وهي طريقة عدد سطور شفيرة المصدر وطريقة نقاط الوظيفة وطريقـــة نقاط الميزة وطريقة نقاط الهدف .

– الإنتاجية: وهي مقياس لتقويم كفاءة المبرمجين المشاركين في فريـــق تطـــوير المشاريع ، ونتناول هنا مقياس (LOC) كمثال ونبين فيه آراء المناصـــرين لهـــذا المقياس ومعارضيه

– منهجيات تقدير الجهد وفيها نتعرف على المعادلة العامة التي يقدر بها الجهد وهي :

المجهود = حجم العمل / معدل الإنتاجية ، وسنتعرف على صعوبات تقدير الجهد اللازم لتطوير المشاريع.

– تقدير التكلفة وسنبحث فيه ثلاثة متغيرات أساسية تدخل في تقدير تكلفة المشروع البرمجي وهي

تكلفة العتاد و تكلفة السفر للعاملين والتدريب والتكاليف المدفوعة للمتطورين.

– تقدير فترة الجدولة الزمنية وفيه نتناول أقسام التقديرات الخاصة، وسنبحث عن مواصفات التقدير الجيد.

– تختتم الوحدة بأفكار مفيدة لتقدير تكلفة المشروعات.

تجد عزيزي الدارس فى ثنايا الوحدة بعض التدريبات التي يساعدك تنفيذها على فهم محتوى المادة هذا بالإضافة إلى أسئلة التقويم الذاتي التي تهدف إلى ترسيخ الفهم وتعزيزه لديك و تعميقه .

أهداف الوحدة

بنهاية دراسة هذه الوحدة ينبغي أن تكون قادراً على أن :

- تعرِّف عملية تقدير تكلفة المشاريع البرمجية.
- تشرح أهمية التقدير الدقيق لتكلفة المشاريع البرمجية.
- تعدد العوامل الرئيسة التي تتحكم في عملية تقدير تكلفة المشاريع البرمجية.
- تعدد أهم العوامل التي تؤثر على دقة تقدير تكلفة المشاريع البرمجية.
- تذكر أهم الأسئلة التي يجب الإجابة عليها أثناء تقدير تكلفة المشاريع البرمجية.
- يُقدر حجم أي برمجية كدالة في عدد أسطر الشيفرة، أو كدالة في عدد نقاط الوظيفة ، أو عدد نقاط الميزة، أو عدد نقاط الهدف.
- تقدر إنتاجية أعضاء فريق تطوير المشاريع البرمجية.
- تشرح أهم المنهجيات المستخدمة في تقدير الجهد المطلوب لتطوير المشاريع البرمجية.
- تستخدم بعض منهجيات تقدير الجهد في تقدير الجهد المطلوب لتطوير مشروع برمجي .
- تقدر فترة الجدولة الزمنية للمشروعات البرمجية.
- تعدد أهم مواصفات التقدير الجيد للمشاريع البرمجية.
- تتبنى أفكار مفيدة لتقدير تكلفة المشاريع البرمجية.
- تصف النماذج الخوارزمية المستخدمة في تقدير تكلفة المشاريع البرمجية.
- تعدد أهم الصفات المشتركة بين النماذج الخوارزمية المختلفة.
- تقيس مدى دقة تقدير تكلفة المشاريع البرمجية لأي نموذج خوارزمي.

تقوم نماذج تقدير تكلفة المشاريع البرمجية من خلال استخدام أهم المعايير الخاصة بذلك.

تمهيد

في عصرنا الحالي أصبحت تكلفة المشروع البرمجي هي العنصر الأساسي في تكلفة معظم الأنظمة المعتمدة على الحاسب الآلي ، وتُعرف بأنها عملية التنبؤ بالمجهودات والموارد المطلوبة لتطوير المشروع البرمجي والتحقق والمصادقة على صحته وإدارة النشطات المختلفة الخاصة بذلك ، وكذلك تُعد من أهم التحديات والنشاطات المهمة في دورة حياة تطويره ؛ حيث إنه بدون تقدير هذه التكلفة بصورة موثوق بها لا يمكن تخطيط المشروع البرمجي والتحكم في تنفيذه بطريقة فاعلة وصحيحة.

1. أهمية التقدير الدقيق لتكلفة المشاريع البرمجية

Importance of Accurate Cost Estimation of Software Projects :

بصفة عامة ، يتضمن تقدير المشروع البرمجي تقدير المعالم الأساسية للمشروع **(Project Key Milestones)** والحجم **(Size)** والموارد **(Resources)** ، وفريق التطوير **(Staffing)** والجداول الزمنية **(Schedules)** ، التكلفة **(Cost)** ، ويتم ذلك من خلال تنفيذ عدد من الخطوات التكرارية **(Iterative Steps)** ، والتي سوف نتناولها بالشرح المفصل في الفصل الثالث من هذه الوحدة.

يتحكم في تطوير المشاريع البرمجية أربعة عوامل رئيسة بصورة نموذجية هي : الوقت **(Time)** ، والمتطلبات **(Requirements)** ، والموارد **(Resources)** (فريق العمل ، البنية التحتية ، والاعتمادات المالية ، .....) ، والمخاطر **(Risks)** ، وأي تغير غير متوقع في هذه العوامل سوف يؤثر تأثيراً مباشراً على خطة تطوير المشروع.

لذا فعملية التقدير الجيدة والموثوق بها لكل من هذه العوامل تُعد حرجة **(Crucial)** بالنسبة لتطوير المشروع. فالتقدير الأقل من التكلفة الفعلية **(Under-Estimating Cost)** يمكن أن يقود إلى :

- تكوين فريق عمل قليل العدد مقارناً بالمجهود المطلوب لتطوير المشروع ، مما يؤدي إلى إجهاد هذا الفريق ، وبالتالي عدم الالتزام بالجدول الزمني الخاص بتطويره.
- تحقيق مستوى أقل من أفق ضمان الجودة المطلوب ، مما يؤدي إلى تسليم المنتج البرمجي وهو يعمل ضمن منطقة مخاطر الجودة المنخفضة.
- تحديد جدول زمني قصير المدى لإنجاز المشروع ، مما يؤدي إلى عدم تسليمه في المواعيد المحددة ، وبالتالي فقد الثقة في مطور المشروع.

أما التقدير الأعلى من التكلفة الفعلية **(Over-Estimating Cost)** فيمكن أن يؤدي إلى إضافة موارد غير ضرورية وعدم تحكم كافٍ لأفق المشروع ، مما يزيد من مدة تسليم المشروع ، وبالتالي عدم الاستفادة منه في الوقت المناسب ، وربما يؤدي إلى تأجيل تنفيذ المشروع لعدم وجود الموارد المالية اللازمة ، وكذلك ربما يؤدي إلى عدم فوز الشركة أو المؤسسة المطورة بالعقد الخاص بتطوير المشروع مما يؤدي إلى فقدان بعض الوظائف بها.

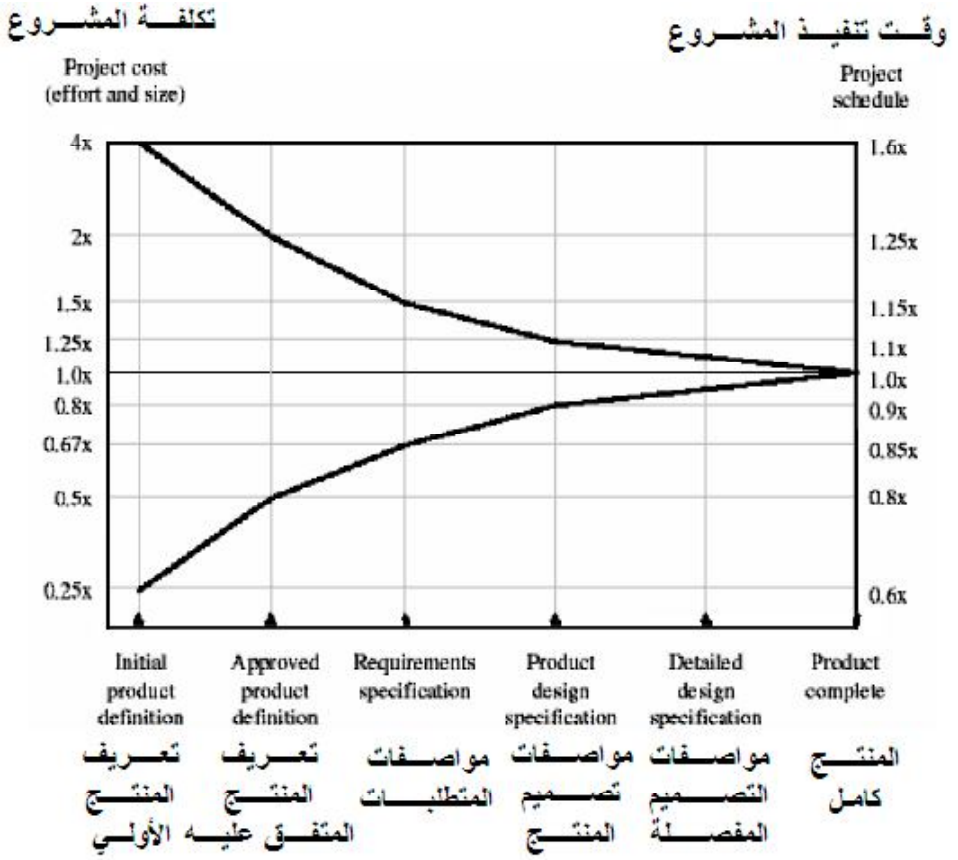
2. العوامل التي تؤثر علي دقة تقدير تكلفة المشاريع البرمجية

Factors That Affect the Accuracy of Software Cost Estimation :

تتوقف دقة تقدير تكلفة المشاريع البرمجية على العديد من العوامل من أهمها :

① درجة دقة تقدير حجم وأفق المشروع.

②وقت عملية التقدير ؛ حيث وجد Boehm في كتابه الشهير "اقتصاديات هندسة البرمجيات" [7] أن عملية تقدير تكلفة المشاريع التي تتم في المراحل الأولي من دورة حياة المشروع تكون بعيدة عن التقدير الدقيق وتصبح أكثر دقة كلما تقدمت عملية التطوير ، ويمكن معرفتها على وجه الدقة عندما ينتهي العمل به ، وذلك كما هو موضح بالشكل رقم 6.1



شكل رقم 7.1 : شكل توضيحي لدقة التقدير في المراحل المختلفة لتطوير المنتج البرمجي ([8] عن[7] و [9])

وكما هو واضح من الشكل رقم 6.1 إنه في الوقت المفترض أن يقدر فيه تكلفة المشروع البرمجي ، وهي مرحلة التعريف الأولى للمنتج (Initial Product Definition Stage) ، يمكن أن يقدر الفرق بين أعلى تقدير وأقل تقدير بـ 16 مرة ، وحتى بعد الانتهاء من مرحلة إعداد مواصفات المنتج (Requirements Specification) يكون الجهد المقدر في حدود 50% ، ويقل الفرق بين أعلى تقدير وأقل تقدير حتى يتساوى تقريباً مع تسليم المنتج النهائي.

جدول 6.1 : عوامل ضرب التقدير للمراحل المختلفة لتطوير المشروع [8]
(Estimation Multiplier by Project Phase)

المرحلة (Phase)	عامل ضرب التكلفة (الحجم والمجهود) Cost (Size and Effort) Multiplier		عامل ضرب فترة التطوير Schedule Multiplier	
	Optimistic تفاؤلي	Pessimistic تشاؤمي	Optimistic تفاؤلي	Pessimistic تشاؤمي
التعريف الأولي للمنتج	0.25	4.0	0.60	1.6
التعريف المتفق عليه للمنتج	0.50	2.0	0.80	1.25
مواصفات المتطلبات	0.67	1.5	0.85	1.15
مواصفات تصميم المنتج	0.8	1.25	0.90	1.10
مواصفات تصميم المنتج المفصلة	0.9	1.10	0.95	1.05

والجدول رقم 6.1 يوضح مضمون الشكل 6.1 بصورة جدولية ، حيث يبين عوامل الضرب **(Multipliers)** للحصول على : التقديرات المتفائلة **(Optimistic Estimates)** ، والتقديرات المتشائمة **(Pessimistic Estimates)** لكل من الجهد **(Effort)** والحجم **(Size)** وفترة الجدولة الزمنية **(Schedule)** ، وذلك عبر دورة حياة المنتج.

❶ دقة تحديد متطلبات المشروع ومقدار ثباتها أثناء عملية التطوير.

❷ مقدار ثبات ظروف بيئة التطوير التي على أساسها تم تحديد متطلبات المشروع.

❸ دقة ترجمة هذا الحجم وهذه المتطلبات إلى جهد بشري ، وخطة زمنية، ونفقات مالية.

❹ قدرات وكفاءات وإنتاجية فريق العمل ومدى توافقها مع خطة المشروع.

❺ حجم البرمجيات الجاهزة التي يمكن إعادة استخدامها في المشروع الجديد.

❻ مدى دراية وخبرة مديري المشاريع وطاقم التطوير بالتغير الهائل في التقنيات المستخدمة في المشاريع ، حيث إن هناك طرق تطوير جديدة تم استحداثها في الآونة الأخيرة ومن أمثلتها :

- استخدام البرمجة الكينونية **(Object-Oriented)** في عملية التطوير بدلا من التطوير المبني على الدوال **(Function-Oriented)**.
- استخدام تطبيقات خادم العميل **(Client-Server Applications)** ، والتطبيقات المعتمدة على الويب **(Web-Based Applications)** بدلا من التطبيقات المبنية على نظم الحاسبات المركزية – **(Mainframe System Based Applications)**.
- استخدام المكونات المتناولة **(Off Shelf Components)** والمُعدة سابقاً لبعض أجزاء المشروع بدلا من كتابتها من جديد.
- تطوير المشاريع البرمجية بطريقة هندسة البرمجيات المبنية على المكونات **(Components-Based Software Engineering)** ، بغرض إعادة الاستخدام لبعض المكونات في مشاريع أخرى.
- استخدام أدوات هندسة البرمجيات بمساعدة الكمبيوتر ومولدات البرامج **(CASE Tools and Programs Generators)** بدلا من الوسائل القديمة.

أسئلة تقويم ذاتي

اذكر أهم العوامل (المخاطر) المترتبة على كل من :

ا) التقدير الأقل من التكلفة الفعلية للمشروع.

ب) التقدير الأعلى من التكلفة الفعلية للمشروع.

ما هي أهم العوامل التي تتوقف عليها دقة تقدير تكلفة المشاريع البرمجية؟

3. من هو المسئول عن تقدير تكلفة المشـــروعات البرمجية؟

Who is Responsible for Software Cost Estimation?

المسئولون عن عملية تقدير تكلفة المشروعات البرمجية يمكن أن يختلفـوا من منشأة إلى أخرى حسب تنظيم وظروف كل منها ، ولكن في معظم المنشـآت البرمجية تتم عملية التقدير كالتالي :

①تكليف فريق تطوير المشروع البرمجي بإنجاز عملية التقدير.

②تكليف مدير المشروع بتقديم تقدير واقعي لتكلفة المشروع البرمجـي ، حيـث يمكنه القيام بهذه المهمة بصورة منفردة أو بمشورة المبرمجين المسـئولين عـن المشروع.

③ تكليف المبرمجين أنفسهم بالقيام بعملية التقدير ، حيث أثبتت معظم الدراسـات أن اشتراك المبرمجين في عملية التقدير تجعلها أكثر دقة.

④ تكليف فريق مستقل عن فريق تطوير المشروع لإنجاز عملية التقدير.

⑤ تكليف خبراء مستقلين من خارج المنشأة للقيام بعمليـة التقـدير ، ومـن ثـم التشاور مع فريق مشكل من المنشأة للوصول إلى التقدير النهائي.

لذا ، ولمحاولة ضمان الدقة في تقدير هذه التكلفة فإنه يجب علـى مـديري المشاريع البحث عن إجابات للأسئلة التالية :

- ما مقدار الجهد **(Effort)** اللازم لاستكمال نشاط بحجـم **(Size)** معـين (فرعية معينة)؟.
- ما هي فترة الجدولة الزمنيـة **(Schedule)** اللازمـة لاسـتكمال هـذا النشاط؟.

- ما هي التكاليف الكلية **(Total Cost)** الخاصة بهذا النشاط؟.
- ما هي النشاطات الإدارية المتداخلة في تقدير كامل للمشروع ؟.
- ما هي إنتاجية كل فرد من فريق العمل في عملية التطوير؟.

أسئلة تقويم ذاتي

من هو المسئول عن تقدير تكلفة المشروعات البرمجية؟ وما هي أهم الأسئلة التي يجب على مديري المشاريع البحث عن إجابات لها لدقة تقدير تكلفة البرمجيات؟

4. طرق تقدير حجم البرمجية

Methods of Software Size Estimation :

بصفة عامة يوجد أربع طرق لتقدير حجم البرمجية **(Software Size)** ، سوف نتناولها فيما يلي :

- **الطريقة الأولى : عدد سطور شيفرة المصدر (Source Lines-of-Code : "SLOC")**

تُعد هذه الطريقة أكثر الطرق انتشاراً في تقدير حجم البرمجية لسهولتها ؛ حيث إنها تعتمد على قياس عدد سطور الشيفرة البرمجية بعد استبعاد سطور التعليقات **(Comments Lines)** ، والسطور الخالية **(Blank Lines)**. نموذجياً يوجد نوعان من هذا المقياس : مقياس عدد السطور الفيزيائية **(Physical Lines)** أو عدد السطور المنطقية **(Logical Lines)** ، فمقياس السطور الفيزيائية سهل القياس فهو عبارة عن عدد السطور المنتهية بمفتاح **Enter** ، أو من خلال رمز يُدخل من قبل المستخدم ليدل على قطع السطر الحالي للشيفرة وبداية سطر جديد **(Hard Line Break)**، أما مقياس السطور المنطقية فهو عبارة عن عدد التعليمات

البرمجية **(Software Instructions)** ، والتي يكون لها بداية ونهاية ، ويمكن للتعليمة البرمجية الواحدة أن تشمل عدداً من السطور الفيزيائية . ففي لغة C ــ مثلاً ــ يتطلب مقياس السطور المنطقية عد **(Counting)** الفصلات المنقوطة **(Semicolons)** ومجموعات أزواج الأقواس **(Sets of Open-Closed Braces)**. في بعض لغات البرمجية يمكن أن يتساوى مقياس السطور الفيزيائية ومقياس السطور المنطقية بصورة تقريبية ، ولكن في البعض الآخر يمكن أن يختلفا اختلافا كبيراً ، لذا فإن مقياس SLOC يعتمد على لغة التطوير ، والجدول رقم 6.2 يبين طريقة التحويل من مقياس SLOC المعتمد على سطور الشيفرة الفيزيائية إلى مقياس SLOC المعتمد على سطور الشيفرة المنطقية لأنواع مختلفة من لغات البرمجة.

جدول 6.2 : تحويل عدد السطور الفيزيائية للشيفرة إلى العدد المكافئ من السطور المنطقية لأنواع مختلفة من لغات البرمجة ([10] عن[11])

اللغة (Language)	لاستنتاج عدد السطور المنطقية للشيفرة To Derive Logical SLOC
لغة التجميع ولغة الفورتران Assembly and Fortran	**افترض أن : Logical SLOC = SLOC**
لغات الجيل الثالث (e.g. COBOL , Pascal , Ada83)	**خفض عدد السطور الفيزيائية بنسبة 30%**
لغات الجيل الرابع (e.g. SQL , Perl, Oracle)	**خفض عدد السطور الفيزيائية بنسبة 40%**
اللغات موجهة الأهداف (e.g. Java, C++, Ada95)	**خفض عدد السطور الفيزيائية بنسبة 30%**

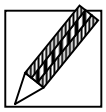
ومعظم نماذج تقدير التكلفة التجارية **(Commercial Cost Estimation Models)** تستخدم مقياس سطور الشيفرة المنطقية لتقدير التكلفة ، وذلك لتقليل اعتماد هذا المقياس على لغة التطوير ؛ حيث إنه لا يعتمد على الشكل الفيزيائي **(Physical Format)** التي تظهر به التعليمات، لذا في حالة الحصول على مقياس

الشيفرة الفيزيائية يجب تحويله أولاً إلى مقياس الشيفرة المنطقية قبل استخدامه ، وخلال الأجزاء الباقية من هذا الكتاب سوف يستخدم الاختصار **LOC** بدلاً من **SLOC** للدلالة على عدد سطور شيفرة المصدر المنطقية.

- **الطريقة الثانية : نقاط الوظيفة ("Function Points "FP") :**

تُعد نقاط الوظيفية **(FP)** مقياساً تركيبياً **(Synthetic Measure)** يعتمد على وظائفية البرمجية **(Software Functionality)** وليس على نوع لغة التطوير ، كما هو الحال في مقياس **LOC** ، ويستخدم ــ غالباً ــ في قياس حجم البرمجية في المراحل الأولية من دورة حياتها وخصوصاً بعد الانتهاء من إعداد مواصفات متطلبات البرمجية **(SRS)** ، حيث إنه من السهل قياس نقاط الوظيفة من خلال تحليل مواصفات المتطلبات الوظيفية للبرمجية ، وخصوصاً في نظم إدارة المعلومات **("Management Information Systems "MISs)** ، في الوقت الذي يكون فيه من الصعب قياس حجم البرمجية باستخدام مقياس **LOC** ، والجدول رقم 7.3 يوضح الفرق بين مقياس **LOC** ومقياس **FP** لتقدير حجم البرمجية.

تدريب (1)



وازن بين مقياس **LOC** ومقياس **FP** لتقدير حجم البرمجية

وقد تم اقتراح مقياس نقاط الوظيفة لأول مرة من قبل **Albrecht** .

وعدد نقاط الوظيفة **(Function Points)** ــ كما عرفها **Albrecht** ـ تعتمد على عدد ، ونوع ، ومدي تعقيد خصائص البرمجية التالية :

- **المدخلات (Inputs)** : وهي تشير إلى أنواع المدخلات المختلفة مثل : الشاشات **(Screens)** والنماذج **(Forms)** ومربعات الحوار **(Dialog Boxes)** وما شابه ذلك ، وكذلك إشارات التحكم **(Control Signals)** التي من خلالها يقوم المستخدم النهائي أو برنامج آخر بإضافة ، أو بحذف ، أو بتعديل بيانات البرنامج.
- **المخرجات (Outputs Types)** : وهي تشير إلى أنواع المخرجات المختلفة مثل : الشاشات **(Screens)** والتقارير **(Reports)** والرسومات **(Graphs)** ، رسائل الأخطاء **(Error Messages)** ، وما شابه ذلك ، وكذلك إشارات التحكم الذي يقوم البرنامج بتوليدها لاستخدامها من قبل المستخدم النهائي أو برنامج آخر.
- **الاستفسارات (Inquiries)** : والاستفسار هو عبارة عن استرجاع بيانات من قاعدة البيانات **(Database)** عن طريق بحث مقيد بشروط محددة يجب توافرها في هذه البيانات ، ويمكن تعريفه بأنه دخل فوري **(Immediate Input)** يؤدي إلى استجابة فورية من البرنامج على شكل خرج فوري **(Immediate Output)** بشكل مبسط.
- **الملفات الداخلية المنطقية (Logical Internal Files)** : وهي تشمل جميع الملفات المنطقية الرئيسة ، والتي يمثل كل منها تجميع منطقي لمجموعات من البيانات التي قد تكون جزءاً من قاعدة بيانات واسعة (مثل جدول في قاعدة بيانات علائقية) ، أو جزءاً من ملف مستقل **(Single Flat File)**.
- **ملفات المواجهة الخارجية (External Interface Files)** : وتشمل جميع الملفات التي تستخدم لنقل أو لمشاركة البيانات مع نظام آخر.

حيث يتم تقويم كل من هذه الخصائص وإعطاء وزن **(Weight)** لكل منها يتراوح من 3 نقاط لكل من المدخلات البسيطة إلى 15 نقطة للملفات الداخلية

المعقدة ، وتقدر نقاط الوزن وفقا للخبرة ، كما هو موضح بالجدول رقم 6.3.

جدول 6.3 : عوامل ضرب نقاط الوظيفة (Function Points Multipliers) [1 , 13]

خصائص البرنامج **Program Characteristics**	مدى تعقيد نقاط الوظيفة **Function Points Complexity**		
	منخفضة Low	**متوسطة Medium**	**عالية التعقيد High**
عدد المدخلات No. of Inputs	**X 3**	**X 4**	**X 6**
عدد المخرجات No. of Outputs	**X 4**	**X 5**	**X 7**
عدد الاستفسارات No. of Inquiries	**X 3**	**X 4**	**X 6**
عدد الملفات الداخلية المنطقية No. of Logical Internal Files	**X 7**	**X 10**	**X 15**
عدد ملفات المواجهة الخارجية No. of Interface Files	**X 5**	**X 7**	**X 10**

ونقـــاط الوظيفـــة غيـــر المعدلـــة **"UFC" (Unadjusted Function-Point Count)** يمكن حسابها بضرب عدد كل خاصية بالوزن المقدر له وجمـــع جميـــع القيم ، كالتالي :

$$UFC = \sum_{i=1}^{5} \sum_{j=1}^{3} N_{ij} W_{ij}$$

حيث Nij ، Wij يمثلان على التـــوالي عـــدد **(Number)** ووزن **(Weight)** أنـــواع الخاصية "i" بتعقيد من الدرجة "j" . وتقوم المؤسسات والشركات التـــي تســـتخدم منهجية نقاط الوظيفة في تقدير حجم البرمجية بتطوير معايير لتحديد درجة تعقيـــد

الخصائص السابقة ، ومع ذلك فإن تحديد درجة التعقيد تعتمد على خبرة المـــثمن بالدرجة الأولى.

مثـال : يبـين الجـدول رقـم 6.4 تقـديرات خصـائص برمجيـة مـا ومـدى تعقيـداتها ، أحسب عدد نقاط الوظيفة غير المعدلة المقابلة لذلك.

جدول 6.4 : مثال على حساب عدد نقاط الوظيفة

خصائص البرنامج **Program Characteristics**	مدى تعقيد نقاط الوظيفة **Function Points Complexity**		
	منخفضة Low	**متوسطة Medium**	**عالية التعقيد High**
عدد المدخلات No. of Inputs	**6X3=18**	**2X4=8**	**3X6=18**
عدد المخرجات No. of Outputs	**7X4=28**	**7X5=35**	**0X7=0**
عدد الاستفسارات No. of Inquiries	**0X3=0**	**2X4=8**	**4X6=24**
عدد الملفات الداخلية المنطقية No. of Logical Internal Files	**5X7=35**	**2X10=20**	**3X15=45**
عدد ملفات المواجهة الخارجية No. of Interface Files	**9X5=45**	**0X7=0**	**2X10=20**
نقاط الوظيفة غير المعدلة (UFC)			304
مضروب التأثير (Influence Multiplier)			1.15
مجموع نقاط الوظيفة المعدلة (Total Adjusted Function Points)			350

$$UFC = (N_{11}W_{11}+N_{12}W_{12}+N_{13}W_{13}) + (N_{21}W_{21}+N_{22}W_{22}+N_{23}W_{23}) + (N_{31}W_{31}+N_{32}W_{32}+N_{33}W_{33}) + (N_{41}W_{41}+N_{42}W_{42}+N_{43}W_{43}) + (N_{51}W_{51}+N_{52}W_{52}+N_{53}W_{53})$$

$$= (6*3+2*4+3*6) + (7*4+7*5+0*7) + (0*3+2*4+4*6) +$$
$$(5*7+2*10+3*15) + (9*5+0*7+2*10) = 304$$

وعدد نقاط الوظيفة النهائية للبرمجيـــة يجـــب أن يـــتم تعـــديلها بمضـــروب التأثير **(Influence Multiplier)** لتعكس مدى درجة تعقيد المشروع ككل ، حيـــث يتم حسابه من المعادلة التالية :

$$\textbf{Influence Multiplier} = 0.65 + 0.01 \sum_{i=1}^{14} Fi$$

حيث F_i هي قيم تعديل درجة التعقيد ، وتستند إلى الإجابات عن الأسئلة الموضحة بالجدول رقم 6.5 (كما اقترحهـــا **L. J. Arthur** ، وترجمهـــا مركـــز التعريـــب والبرمجة بالدار العربية للعلوم عن روجر بريسمان) ، والتي يقدر عددها بأربعـــة عشر سؤال. وبعد ذلك يتم حساب نقاط الوظيفة النهائية بضرب عدد نقاط الوظيفـــة غير المعدلة **"UFC"** في مضروب التأثير طبقاً للمعادلة التالية :

$$FP = UFC * (0.65 + 0.01 \sum_{i=1}^{14} Fi)$$

مع ملاحظة أن أقصى قيمة لمجموع قيم تعديل درجة التعقيد هي (70) ، حيث إن قيمة كل عامل من عوامل التعقيد الأربعة عشر تتراوح من (0) إلى (5) ، ويـــتم حسابها بتقدير كل سؤال (عامل) بقيمة وزن من صفر إلى خمسة طبقـــاً لأهميتـــه وتأثيره في المشروع ، وذلك كما يلي : عامل أساسي (5) ، عامـــل مهـــم (4) ، عامل متوسط (3) ، عامل معتدل (2) ، عامل عرضي (1) ، عامل بدون تـــأثير (0) ، كما هو موضح بالجدول رقم 6.5 ، وبالتالي تتراوح قيمة مضروب التـــأثير من (0.65) كقيمة دنيا إلى (1.35) كقيمة قصوى. والقيم المدونة بالجدول هـــي تكمله للمثال السابق لحساب قيمة مضروب التأثير.

جدول رقم 6.5 : **حساب مضروب التأثير**

م	السؤال	الإجابة (قيمة الوزن)
1	هل يتطلب النظام عملية نسخ احتياطي (Backup) ، واستعادة (Recovery) دورية يمكن الوثوق بها؟.	5
2	هل هناك حاجة إلى اتصالات البيانات؟.	2
3	هل هناك وظائف معالجة موزعة؟.	0
4	هل فعالية الأداء أمر حرج؟.	5
5	هل سيعمل النظام ضمن بيئة تشغيلية قائمة وكثيرة الاستخدام؟.	3
6	هل يتطلب النظام إدخالاً أنياً للبيانات؟.	4
7	هل يتطلب إدخال البيانات المباشر أن تبنى معاملات الدخل عبر شاشات أو عمليات متعددة؟.	3
8	هل يتم تحديث الملف الرئيس تحديثاً مباشراً؟.	3
9	هل المدخلات والمخرجات أو الملفات أو الاستفسارات معقدة؟.	4
10	هل المعالجة الداخلية معقدة؟.	3
11	هل تم تصميم الشيفرة بحيث يمكن إعادة استخدامها؟.	3
12	هل يتضمن التصميم عمليتي التحويل (Conversion) والتثبيت (Installation)؟.	5
13	هل تم تصميم النظام بحيث يمكن تثبيته تثبيتاً متعدداً في مؤسسات مختلفة؟.	5
14	هل تم تصميم التطبيق بحيث يسهل تغييره ويوفر سهولة استعمال للمستخدم؟.	5
	إجمالي قيم تعديل درجة التعقيد	50

Influence Multiplier = 0.65 + 0.01 (50) = 1.15

FP = 304 * 1.15 = 350

ويمكن استخدام عدد نقاط الوظيفة لتقدير الحجم النهائي للشيفرة باستخدام التحليل التاريخي للبيانات **(Historical Data Analysis)** حيث يمكن حساب المتوسط الحسابي لعدد السطور للغة معينة **(AVC)** اللازم لحساب نقطة دالة ، ومنه يمكن حساب حجم الشيفرة من خلال المعادلة التالية :

حجم الشيفرة = AVC * عدد نقاط الوظيفة

ويوفر الجدول رقم 6.6 تقديرات تقريبية لمعدل عدد أسطر الشيفرة اللازمة لبناء نقطة دالة واحدة لأنواع مختلفة من لغات البرمجة.

جدول رقم 6.6 : تقديرات تقريبية لمعدل عدد أسطر الشيفرة اللازمة لبناء نقطة دالة واحدة لأنواع مختلفة من لغات البرمجة ([1] عن [14] , [13])

لغة البرمجة	AVC	لغة البرمجة	AVC
لغة التجميع "Assembly Language"	320	اللغات الكائنية المنحى "OOPL"	53
اللغة "C"	128	اللغة Ada 9x	49
كوبل "COBOL"	105	لغات الجيل الرابع "4GLs"	20
فورتران "FORTRAN"	106	مولدات الشيفرة "Code Generators"	15
باسكال "PASCAL"	90	SQL	12
اللغة Ada 83	70		

- الطريقة الثالثة : نقاط الميزة **(Feature Points)** :

تم تطوير هذه الطريقة لتقدير أو قياس حجم النظم البرمجية التي تعمل في الوقت الحقيقي **(Real Time System Software)** والتي تعتمد في تكوينها الداخلي

على مجموعة معقدة من الخوارزميات **(High Algorithmic Complexity)** وتملك عدداً أقل من المدخلات والمخرجات **(Few Input/Output)** مقارنة بنظم إدارة المعلومات **(MISs)** ، مثل : البرمجيات الرياضية **(Mathematical Software)** وبرمجيات المحاكاة الرقمية **(Digital Simulation Software)** ، وبرمجيات التطبيقات الحربية **(Military Applications Software)** . وهي تُعد توسعة لمنهجية نقاط الوظيفة لتشمل تأثير مدى تعقيد الخوارزميات على حجم البرمجية ؛ حيث إنه بالإضافة إلى الخمس خصائص (العوامل) المذكورة سابقاً في منهجية نقاط الوظيفة تم إضافة عامل الخوارزمية **(Algorithm Parameter)** وإعطائه عامل وزن (ثقل) افتراضي يساوي (3) ، مع اعتبار عامل الملفات الداخلية المنطقية **(Logical Internal Files)** ذات تعقيد بسيط في مثل هذه الأنظمة وإعطائها عامل وزن افتراضي يساوي (7) بدلاً من (10) للتعقيدات المتوسطة و(15) للتعقيدات العالية. يجب ملاحظة أن نقاط الوظيفة ونقاط الميزة تمثل الشيء نفسه (الوظيفة الناتجة عن البرمجية) ، وبالتالي فأنه في التطبيقات البرمجية التي تتساوي فيها عدد الخوارزميات وعدد الملفات الداخلية المنطقية ، أو عندما لا يؤخذ في الاعتبار إلا بُعد البيانات لتطبيق ما فإنه تتساوي قيم نقاط الوظيفة ونقاط الميزة ، أما في التطبيقات البرمجية ذات الكثافة الخوارزمية (معظم أنظمة الزمن الحقيقي الأكثر تعقيداً) فإن عدد نقاط الميزة تزيد عن عدد نقاط الوظيفة ، والجدول رقم 6.7 يبين النسبة بين عدد نقاط الميزة إلى عدد نقاط الوظيفة لمجموعة من التطبيقات البرمجية المختلفة.

جدول رقم 7 .6 : النسبة بين عدد نقاط الميزة إلى عدد نقاط الوظيفة لمجموعة من التطبيقات البرمجية المختلفة **[15 عن 13]**

نقاط الميزة Feature Points	نقاط الوظيفة Function Points	نوع التطبيق Application Types
0.80	1	Batch MIS Projects
1.00	1	On-Line MIS Projects
1.00	1	On-Line Database Projects
1.20	1	Switching System Projects
1.35	1	Embedded Real Time Projects
1.50	1	Factory Automation Projects
1.75	1	Diagnostic and Prediction Projects

- الطريقة الرابعة : نقاط الهدف **(Object Points)** :

في الآونة الأخيرة تم استخدام مقياس آخر وهو ما يعرف بنقاط الهدف **(Object Points)** كبديل قوي للمقياس السابق (نقاط الوظيفة) حيث استخدمت لغات الجيل الرابع (4GLS) لتطوير البرمجيات ، وقد استخدمت نقاط الهدف في تطوير ما يعرف بموديل (COCOM2) والذي سيتم مناقشته لاحقا.

ونقاط الهدف هذه ليست النقاط المستخدمة في لغة **(Object Oriented)** ولكن عدد نقاط الهدف لبرنامج عبارة عن تقدير :

1. لعدد الشاشات : الشاشة الصغيرة لها (1) نقطة ، الشاشات المعقدة لها (2) نقطة ، والشاشات الأكثر تعقيدا لها (3) نقاط.
2. لعدد التقارير المنتجة : على سبيل المثال التقارير البسيطة لها (2) نقطه ، والمتوسطة لها (5) نقاط والأكثر تعقيدا لها (8) نقاط.
3. لعدد مكونات لغات الجيل الثالث (3GL) : والتي لابد من تطويرها لإتمام شيفرة من الجيل الرابع (4GL) ، ويقدر وزن كل مكون منها بـــ (10) نقاط.

والميزة الأساسية لاستخدام نقاط الهدف بدلا من نقاط الوظيفة أنها سهلة التقدير من ناحية اللغات عالية المستوى حيث ترتبط نقاط الهدف ارتباطاً مطلقاً كما ذكرنا بتطوير الشاشات والتقارير ، ومكونات **(3GL)**.

5. الإنتاجية (Productivity)

الإنتاجية هي : مقياس لتقويم كفاءة المبرمجين المشاركين في فريق تطوير المشاريع البرمجية ، ولا بد من تقديرها من قبل مديري المشاريع من خلال قياس بعض سمات البرمجية ومقارنته بالجهد الكلي المطلوب للتطوير ، وهناك نوعان من المقاييس أو السمات المستخدمة :

❶ المقياس المعتمد على الحجم : وهو يقيس حجم الخرج لنشاط معين ويعتبر عدد الخطوط مقياس يعتمد على الحجم ، ويعد مقاس **(LOC)** مثالاً لهذا النوع.

ويدعي مناصرو مقياس **(LOC)** أنه نتاج إنساني في جميع مشاريع تطوير البرمجيات يمكن إحصاؤه بسهولة ، وأن العديد من نماذج التقدير الحالية تستخدمه دخلاً رئيسياً ، وأنه يوجد الآن حجم كبير من المؤلفات والبيانات التي تسند إليه.

وفي الجانب الآخر يدعي المعارضون أن مقياس **(LOC)** واسع التعريف لقياس الإنتاجية ، ويعتمد على لغة البرمجة ، وأنها تعاقب البرامج المصممة تصميماً جيداً ولكنها أقصر ، وأنه كان سهل الاستخدام حينما كان يتم تطوير البرمجيات باستخدام لغات مثل الفورتران والكوبل ؛ حيث كان كل سطر يتم وضعه في كارت معين في حين أن الوضع الآن قد تغير ؛ ولا يمكن الأخذ بهذا المنطق عند تطوير برمجية بلغة مثل لغة السي أو الجافا والتي يحوي فيها السطر الواحد على عدد من الجمل ، مما جعل العلاقة بين جمل البرامج وعدد السطور

علاقة شائكة وصعبة في مثل هذه اللغات. لذا فإن مقارنة الإنتاجية عبر اللغات المختلفة يمكن أن يعطي انطباعاً خاطئاً عنها فكلما زادت الجمل في اللغة (كلما زاد مستوى اللغة) كلما قلت الإنتاجية ؛ حيث تأخذ نفس الوظائف شفرة أقل لتنجز في لغة المستوى الأعلى منها في لغة المستوى المتدني. فعلى سبيل المثال يمكن تطوير نظام بحجم 5000 سطر بلغة التجميع خلال (28) أسبوع بإنتاجية (714) خط في الشهر ، بينما يمكن تطوير نفس النظام بحجم 1500 سطر بلغة عالية المستوى خلال (20) أسبوع بإنتاجية (300) خط في الشهر، ويتضح من هذا المثال أن الإنتاجية بلغة عالية المستوى تقريبا هي النصف في حالة البرمجة بلغة التجميع ، إضافة إلى أن التكاليف بلغة عالية المستوى أقل وتتم في وقت أقل.

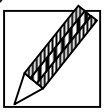
وإضافة إلى ذلك ، يدّعي المعارضون أن مقياس **(LOC)** يتطلب مستوى من التفصيل قد يصعب تحقيقه (أي يجب على المخطط أن يقدر عدد أسطر الشيفرة الواجب إنتاجها قبل اكتمال التحليل والتصميم بوقتٍ طويل).

❷ مقياس يرتبط بالأداء : وهو يرتبط بالأداء الكلي للبرمجية حيث يعبر عن الإنتاجية بعدد الدوال المنتجة في وقت معين وتعتبر نقاط الوظيفة ونقاط الهدف أمثلة من هذا النوع.

ومقياس نقاط الوظيفة بتوسعاتها المختلفة ، مثله مثل مقياس (LOC) ، يُعد مثيراً للجدل ؛ حيث يدعي المناصرون له أنه مستقل عن لغة البرمجة ، وهذا يجعله مثالياً للتطبيقات التي تستخدم اللغات التقليدية والإجرائية ، وأنه يستند إلى بيانات تكون معروفة في مرحلة مبكرة من بدء المشروع ، وهذا يجعله أكثر جاذبية كطريقة تقدير، ويدعي المعارضون له أنه يحتاج إلى براعة المثمنين ؛ حيث إن معظم حساباته تستند إلى بيانات شخصية (Subjective) عوضاً عن

بيانات موضوعية (Objective) ، وأنه يتعذر تجميع خصائص البرمجية والعوامل الأخرى ذات الصلة بعملية التطوير ، وأن عدد نقاط الوظيفة ليس له معنى فيزيائي مباشر ، فما هو إلا رقم.
وإنتاجية المبرمجين لمؤسسة معينة ترتبط ارتباطاً وثيقاً بعدة عوامل مؤثرة.

تدريب (2)



عدد العوامل المؤثرة على إنتاجية المبرمجين

وقد وجد في بعض الحالات أنه في فرق العمل الكبيرة هناك تفاوت كبير في الإنتاجية من مبرمج لآخر يصل مداه (1– 10) ، وفي هذه الحالة لا بد من اللجوء إلى المتوسط الحسابي للإنتاجية ، بينما تعتمد الإنتاجية في فرق العمل الصغيرة إلى حد كبير على المواهب والقدرات الفردية لمكونات هذا الفريق الصغير.

وتعريف المتوسط الحسابي للإنتاجية غير موجود إطلاقا في المؤسسات ، ولكن يمكن تقديرها ما بين (200 ــ 800) سطر/شخص في الشهر **(LOC/P-M)** للتطبيقات التجارية **(Commercial Applications)** ، وما بين (150 ــ 400) سطر/شخص في الشهر **(LOC/P-M)** لبرمجيات النظم **(System Software)** ، وما بين (40 ــ 160) سطر/شخص في الشهر **(LOC/P-M)** لنظم الوقت الحقيقي المتضمنة **(Real Time Embedded Systems)**. ويمكن تقديرها بدلالة نقاط الهدف ما بين (4 ــ 50) نقطة هدف/شخص في الشهر **(Object Point/P-M)**.

أسئلة تقويم ذاتي

ما هي أهم المقاييس التي تستخدم لقياس الإنتاجية؟

6. منهجيات تقدير الجهد

Effort Estimation Approaches

يُعرف المصطلح "الجهد" **(Effort)** بأنه كمية العمل الإنساني **(Human Work Amount)** المبذولة لتطوير المشروع ، ويمكن أن يعبر عنها باستخدام العديد من الوحدات ، مثل : شخص ــ ساعات **(Person - Hours)** ، شخص ــ أيام **(Person- day)** ، شخص ــ أسابيع **(Person - Weeks)** ، شخص ــ شهور **(Person - Months)** ، شخص ــ سنين **(Person - Years)**. ويجب أخذ نظام العمل (عدد أيام العمل في الأسبوع ، عدد ساعات العمل في اليوم ، عدد أيام الأجازات الاعتيادية والمرضية في السنة) في الاعتبار عند حساب المجهود ، فمثلاً في الولايات المتحدة الأمريكية متوسط عدد أيام العمل في السنة حوالي 220 يوم ، 5 أيام عمل في الأسبوع ، 8 ساعات عمل في اليوم (تحسب عادة 6 ساعات بعد استقطاع أوقات الراحة والاجتماعات).

وبصفة عامة يقدر المجهود من خلال المعادلة العامة :

Staff Effort = Size of Work / Production Rate

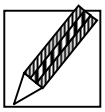
المجهود = حجم العمل / معدل الإنتاجية ، فمثلاً :

عدد أوراق المواصفات / معدل إنتاجية المحلل شهرياً = محلل ــ شهور

عدد سطور الشيفرة / معدل إنتاجية المبرمج شهرياً = مبرمج ــ شهور

ويمكن القول بأنه ليس هناك منهجية بسيطة لعملية تقدير الجهد اللازم لتطوير مشروع برمجي ، حيث يوجد العديد من الصعوبات التي تواجها .

تدريب (3)



من الصعب إجراء تقدير دقيق للمشروع البرمجي في مراحله الأولى علل لذلك

وعموما هناك عدة منهجيات تُستخدم لتقدير الجهد والتكاليف ، والتي يمكن تلخيصها كما أوردها **Boehm (1981)** في التالي :

❶ نمذجة التكلفة الخوارزمية **(Algorithmic Cost Modeling)** : وهي منهجية تعتمد على المعادلات **(Formulaic Approach)** ، حيث إنه يتم تطوير نموذج مبنى على معلومات التكاليف التاريخية **(Historical Information Cost Model)** والتي تربط بعض سمات المشروع البرمجي ــ غالبا ما تكون حجم المشروع ــ بتكاليف المشروع ، ومن ثم عمل تقدير لهذه السمة بحيث يستطيع النموذج المطور التنبؤ بالجهد المطلوب. تمتاز هذه المنهجية بأنها موضوعية (Objective) وذات علاقة مباشرة بعملية التكلفة ، وقابلة للتكرار **(Repeatable)** والتحليل **(Analyzable)** ، وتمتاز بالكفاءة في تحليل الحساسية **(Sensitivity Analysis)** ، ومعايرة بطريق موضوعية للخبرات **(Objectively Calibrated to Experience)** ، ولكن يعيبها الحساسية الزائدة للمدخلات ، حيث إنها ترتبط بمدخلات غير موضوعية **(Subjective Inputs)** ، ومخرجاتها تتأثر بالظروف المحيطة بعملية التطوير.

❷ رأي الخبير **(Expert Judgment)** : وفيه يتم الاستفادة من الخبراء المتخصصين في مجال تطوير المشاريع البرمجية حيث يقوم كل منهم بعمل تقدير لتكلفة المشروع حيث يتم مناقشتها والوصول إلى رأي نهائي متفق عليه لهذه التكاليف ، وتعتمد دقة تقدير هذه التكلفة على مدى دراية هؤلاء الخبراء بمشاريع

مماثلة.

❸ التقدير المبني على التشابه **(Estimation by Analogy)** : وفيه يتم تقدير تكلفة المشروع بناءً على تكلفة مشاريع مماثلة سابقة في مجال التطبيق نفسه ، حيث يتم تقدير تكلفة المشروع قيد الدراسة بالتشابه ، وتمتاز هذه الطريقة بالدقة إذا توافرت بيانات لمشاريع مماثلة للمشروع تحت التقدير ، ويعيبها أنها مستحيلة الحدوث إذا لم يكن هناك مشاريع مماثلة.

❹ قانون باركنسون **(Parkinson's Law)** : ينص على : تمديد المشروع لينفذ خلال الفترة المتاحة ، وهذا يعني أن تقدير التكاليف يتم بناءً على المدة المحددة لتنفيذ المشروع والموارد المتاحة ، فعلى سبيل المثال إذا ما طلب تسليم مشروع برمجي خلال 12 شهرا ولدينا خمس مبرمجين فإن الجهد اللازم يتم تقديره بـــ 60 شخص ــ شهور **(60 person-months)** .

❺ التقدير للفوز **(Pricing to Win)** : وفيه يتم تقدير تكلفة المشروع البرمجي بناءً على مقدرة العميل على الإنفاق عليه ، أي أن الجهد يقدر بناءً على ميزانية العميل وليس على أداء البرمجية. وقد تبدو هذه الطريقة غير أخلاقية ، ولكن يلجأ إليها بعض الشركات والمؤسسات كثيراً في حالة عدم توافر مواصفات تفصيلية للمشروع المطلوب تنفيذه ، وتُقدر التكاليف بحيث تتوافق مع أساسيات العرض العام المقدم بدون النظر إلى جودة أداء المشروع التي يجب أن يتصف بها ، حيث تتقيد عملية التطوير بالتكلفة ، وربما يتم التفاوض بشأن تفاصيل المواصفات أو منهجية الارتقاء بالنظام فيما بعد.

وعموما فإن سريان تقدير الجهد في الطرق السابقة يمكن أن يتم عن طريق اتجاهين إما من "أعلى إلى أسفل" **(Top-Down Approach)** أو من "أسفل

إلى أعلى" **(Bottom-Up Approach)**. والاتجاه الأول "من أعلى إلى أسفل" يبدأ من مستوى النظام ككل حيث يقوم المثمن بفحص الأداء الكلي للنظام وتفريع هذا الأداء إلى فرعيات أقل حيث يؤخذ في الاعتبار تكلفة أنشطة النظام مثل (التكاملية ، الأداء ، التوثيق....) ، وهو قابل للاستخدام بدون معرفة معمارية النظام ومكوناته. والاتجاه الثاني "من أسفل إلى أعلى"على العكس يبدأ من أسفل من مستوى المكونات البسيطة حيث يتم تفكيك النظام إلى مركبات ويتم تقدير الجهد اللازم لكل مركبة وتجميع هذه التكاليف لتقدير التكاليف الكلية ، وهو قابل للاستخدام عند معرفة معمارية النظام وتعريف مكوناته. والميزات للاتجاه "من أعلى إلى أسفل" تُعد عيوب في الاتجاه "من أسفل إلى أعلى" والعكس. فعلى سبيل المثال الاتجاه الأول "من أعلى إلى أسفل" يمكن استخدامه في تقدير تكلفة المسائل المعقدة المصاحبة للنظام كواجهات الاستخدام ، ولكن قد يصعب تقدير تكلفة مشاكل التقنيات الدنيا الصعبة **(Difficult Low Level Technical Problems)** بالدقة المطلوبة ، حيث إن ذلك يتطلب تصميم النظام بالتفصيل ، في حين أن الاتجاه "من أسفل إلى أعلى" يأخذ في الاعتبار تكلفة هذه المشاكل إلا أنها غالية التكاليف حيث إنه لا بد من وضع تصميم بدائي للنظام ، ولكن في الوقت نفسه قد يكون من الصعب تقدير تكلفة أنشطة النظام مثل (التكاملية ، الأداء ، التوثيق....).

ولكل طريقة تقدير نقاط قوه ونقاط ضعف ، ويحبذ للمشاريع الكبيرة أن يُستخدم العديد من الطرق لتقدير كلفتها ، ومقارنة نتائجها ، فإذا أعطت نتائج متنافرة وغير متفقة فهذا يعني عدم توافر المعلومات الكافية عن تقدير التكلفة ، وبالتالي فإنه لابد من النظر في إمكانية الحصول على معلومات أكثر وإعادة تقدير التكلفة بالطرق المختلفة حتى تتقارب تقديراتها.

أسئلة تقويم ذاتي

اشرح أهم الصعوبات التي تواجه عملية تقـدير الجهـد الـلازم لتطوير مشروع برمجي.

7. تقدير التكلفة Cost Estimation

هناك ثلاثة متغيرات أساسية تدخل في تقدير التكاليف الكلية لمشروع تطوير برمجية:

❶ تكلفة العتاد والبرمجيات وهي تشمل الصيانة.

❷ تكلفة السفر للعاملين والتدريب.

❸ التكاليف المدفوعة للمطورين (يطلق عليها تكلفة المجهود).

ويمكننا القول إن التكاليف السائدة في معظم المشاريع هي "تكلفة الجهـد" ؛ حيث تكون تكلفة العتاد اللازمة بسيطة جدا ، وكذلك فإن تكلفة السفر إلى الأماكن المختلفة التي ينفذ بها المشروع بسيطة نسبيا ، حيث إن وجود وسـائل الاتصـال المختلفة (فاكس ، البريد الإليكتروني ، وسائل الاتصال من بعد) يمكـن أن تقلـل وإلى حد كبير من تكلفة السفر.

وتكاليف الجهد ليست فقط تشمل مرتبات فريق تطوير المشروع البرمجـي (مهندسي هندسة نظم البرمجيات ومهندسي هندسة البرمجيات ، ومهندسي اختبار البرمجيات فريق إدارة ودعم المشروع) حيث تحسب الجهات المطـورة تكلفـة الجهد بدلالة التكاليف الكلية حيث يتم تقدير التكاليف الكلية وقسمتها علـى عـدد

المطورين ولذا فإنها فرعية من الفرعيات التالية :

- تكلفة الإعاشة وتشمل الإنارة والتدفئة للمكاتب.
- تكلفة العمالة المساعدة (المحاسبين ، أفراد السكرتارية ، عمال النظافة ، والعمال الفنيين).
- تكلفة الشبكات والاتصالات.
- تكلفة المشاركة في الأماكن الجماعية للعاملين (المطعم للعاملين ، المكتبة،.).
- تكلفة التأمين الاجتماعي للعاملين ، التأمين الصحي ونحوه.

وبالإضافة إلى العوامل الأساسية السابقة يوجد عوامل أخرى يمكن أن تؤثر في تقدير تكلفة المشروع البرمجي من قبل الجهة المطورة من أهمها

① **اكتساب سابقة أعمال :** ربما تتجه الجهة المطورة إلى تحديد سعر رخيص لأنها تريد أن تتطلع إلى مرحلة مستقبلية في سوق البرمجيات وقبولها لهذا الربح البسيط في مشروع ربما يكون سببا في تحقيق أرباح طائلة في مشاريع أخرى مستقبلية وهذه الخبرة المكتسبة ربما تسمح بتطوير منتجات جديدة.

② **تقدير التكاليف غير المؤكد :** في حالة عدم تأكد الجهة صاحبة المشروع من تقدير التكاليف ربما تزيد من السعر وتحقق ربح أعلى من الربح المعتاد.

③ **بنود تعاقدية :** ربما يرغب العميل من المطور استخدام مصدر البرنامج في مشاريع أخرى وفي هذه الحالة تكون التكاليف أقل بكثير عما لو أمتلك العميل مصدر البرنامج.

④ **صلاحية المتطلبات :** إذا ما تغيرت المتطلبات تخفض المؤسسة السعر حتى

تحصل على العقد وبعد حصولها على العقد تـــزداد التكـــاليف بتغييـــر المتطلبات.

⑤ **الحالة المالية :** المطورون الذين لديهم اعتمادات مالية ضـــعيفة غالبـــا مـــا يقومون بتخفيض السعر للحصول على العقد وفـــي هـــذه الحالـــة مـــن المستحسن الحصول على ربح ولو بسيط بدلا من عدم وجود عمل.

أسئلة تقويم ذاتي

(العتاد والبرمجيات، السفر والتدريب للعاملين، الجهود) عبارة عن أهم ثلاثة متغيرات أساسية تدخل في تقدير التكاليف الكلية لمشروع تطوير برمجية، عدد أهم العوامل الأخرى المؤثرة من قبل الجهـــة المطورة والتي يمكن أن تؤثر في تقدير تكلفة المشروع البرمجي؟

8. تقدير فترة الجدولة الزمنية (Schedule Estimation)

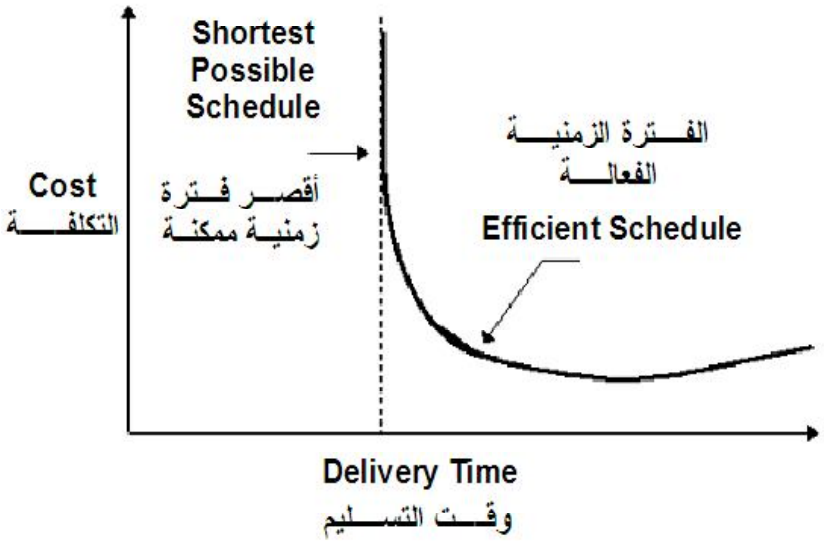
قبل الدخول في كيفية تقدير فترة الجدولة الزمنية **(Schedule)** ، يجـــب أولاً فهم العلاقة بين الجهد **(Effort : Person-Months)** ، وفترة الجدولـــة الزمنيـــة ، والتكلفة **(Cost)**. فالعلاقة بين الجهد وفتـــرة الجدولـــة الزمنيـــة ليســـت خطيـــة (وكذلك بينه وبين التكلفة ، حيث إن العلاقة بين الجهد والتكلفة علاقـــة خطيـــة ، بمعنى إذا زاد الجهد المطلوب لتطوير البرمجية زاد عدد أعضاء فريق التطوير ، وبالتالي تزيد التكلفة وبطريقة خطية) ، كما هـــو موضـــح بالشـــكل رقـــم 6.2 ، والسبب يكمن في إجابتك عن النشاط التالي :

نشاط

إذا كان يمكن لشخص أن يكتب برنامجاً مكوناً من 1000 سطر شيفرة في الأسبوع ، هل يمكن لخمسة أشخاص أن ينتجوا نفس البرنامج في أسبوع ؟، وهل يمكن لأربعين شخصاً أن ينتجوا نفس البرنامج في ساعة؟.

الإجابة بالطبع لا ، ولكن ما هو السبب؟. حاول عزيزي الدارس الإجابة.

يمكن أن تتتاج العديد من العلاقة بين الشكل رقم 6.2 ومنها :



شكل 7.2 : العلاقة بين تكلفة المشاريع البرمجية ووقت التسليم

المصدر [8]

❶ إن تمديد فترة الجدولة الزمنية (فترة تطوير المشروع) بمدة معينة مقبولة يمكن أن يؤدي إلى تخفيض التكلفة بشكل فعال.

❷ يوجد فترة جدولة زمنية لا يمكن أن تتعداها لأي مشروع وبأي وسيلة من الوسائل ، يطلق عليها "أقصر فترة جدولة زمنية ممكنة" **(Shortest Possible Schedule)** ، ويجب عليك معرفتها فقط بدون محاولة تجاوزها.

❸ يمكنك تقصير فترة الجدولة الزمنية للمشروع من خلال :

- خفض وظائف المشروع ، وبالتالي خفض المجهود المطلوب لتطويره ، أو زيادة المطورين بفريق التطوير بشرط أن يعملوا بطريقة متوازية ، أو الحفاظ على عدد المطورين وجعلهم يعملون وقتاً إضافياً.

مع ملاحظة أن زيادة عدد المطورين سوف يزيد من تكلفة المشروع ، وكذلك من العمل المناط بالإدارة ، مع نقص في الإنتاجية نتيجة عملية الاتصالات الزائدة بين المطورين، أما جعلهم يعملون وقتاً إضافياً سوف يزيد من تكلفة المشروع ، ويزيد الإنتاجية في بداية الأمر ولفترة قصيرة ، ثم تقل الإنتاجية تدريجياً نتيجة الإرهاق الذي سوف يصيب أعضاء فريق التطوير نتيجة العمل المستمر.

وتنقسم التقديرات الخاصة بفترة الجدولة الزمنية إلى ثلاثة أنواع :

① النوع الأول : أقصر فترة جدولة زمنية ممكنة **(Shortest Possible Schedules)** :

حيث لا يمكن ضغط فترة الجدولة الزمنية لأصغر من ذلك ، حيث إنها مضغوطة لأكثر درجة ممكنة.

ولتحقيق **"Shortest Possible Schedule"** : يجب أن تتوفر الشروط التالية :

- أن يكون فريق التطوير على درجة عالية جداً من الكفاءة والمهارة والخبرة في مجال تطبيق المشروع ولغة التطوير لعدد كبير من السنين ، وأن يكونوا مختارين من ضمن أعلى 10% من مجمع المطورين الموهوبين.
- أن تكون إدارة المشروع مثالية من حيث التوجيهات للمطورين للعمل ضمن نشاطات المشروع فقط ، وبدء العمل من أول يوم بالمشروع حتى تسليمه.
- أن تكون عمليات التطوير مُعرَّفة ومفهومه جيداً وناضجة ، وأن تتحقق خطوات تطوير المشروع بطريقة مثالية وباستخدام أحدث التقنيات كفاءة من حيث توفير الوقت.
- أن تكون بيئة التطوير مثالية من حيث توافر أدوات البرمجيات الحديثة ، الوصول غير المحدد لجميع أعضاء الفريق لموارد الحاسب الآلي وتوافر جميع وسائل الاتصال مثل الهواتف المباشرة والبريد الصوتي والفاكس وشبكة الحاسب ، والبريد الإلكتروني ، ومؤتمرات الفيديو ، وأن يكون الفريق بأكمله في مكان واحد يتيح الاتصال الفوري بين أعضاء الفريق وأن يكون مجهز تجهيزاً مثالياً من حيث الإضاءة والمكيفات وخلافه.

② النوع الثاني : فترة الجدولة الزمنية الفعالة **(Efficient Schedules)** :

للأسف العديد من المنشأت لا تملك الشروط الكفيلة بتحقيق **"Shortest Possible Schedule"** ، لذا فإنه في حالة ما إذا كانت الأولوية القصوى لمدة تنفيذ المشروع يكون من الأفضل التفكير في تحقيق أقصر فترة جدولة زمنية يمكن الوصول إليها والتي يطلق عليها "فترة الجدولة الزمنية الفعالة" **(Efficient Schedule)** ، بدون التفكير في الوصول إلى **"Shortest Possible Schedule"** ،

ولتحقيق ذلك يجب أن يكون فريق التطوير على درجة من الكفاءة والمهارة الخبرة في مجال تطبيق المشروع ولغة التطوير لعدد محدود من السنين ، ومختارين من ضمن أعلى 25% من مجمع المطورين الموهوبين ، كما أن احتمال انسحاب أعضاء من الفريق يجب أن تكون أقل من 6% في العام ، هذا بالإضافة إلى توفر باقي الشروط المذكورة أعلاه في **"Shortest Possible Schedule"** من حيث الإدارة وبيئة وعمليات التطوير.

③ النوع الثالث : فترة الجدولة الزمنية الظاهرية **(Nominal Schedules)** :

في حالة كون مدة تنفيذ المشروع لا تمثل أولوية ، أو لا يمكن توفير الشروط المطلوب لتحقيق **"Shortest Possible Schedule"** ، أو **"Efficient Schedule"** ، فيمكن التفكير في الوصول إلى أقل جهد ممكن وفترة زمنية مقبولة ، وهو ما يطلق عليه "فترة الجدولة الزمنية الظاهرية" **(Nominal Schedule)** ، ولتحقيق ذلك يجب أن : يكون متوسط أعضاء فريق التطوير على درجة من الكفاءة والمهارة والدراية المحدودة في مجال تطبيق المشروع ولغة التطوير ، وأن يكونو مختارين من ضمن أعلى 50% من مجمع المطورين الموهوبين ، أن يكون احتمال انسحاب أعضاء من الفريق أقل من 12% في العام ، مع توفر باقي الشروط المذكورة أعلاه ولكن بدرجة أقل بكثير مما هي مطلوبة لتنفيذ المشاريع ذات الجدولة الزمنية الفعالة والقصيرة.

أسئلة تقويم ذاتي

ما هي الشروط التي يجب أن تتوفر لتحقيق :

أ) أقصر فترة جدولة زمنية ممكنة.

ب) فترة الجدولة الزمنية الفعالة.

ج) فترة الجدولة الزمنية الظاهرية.

9. مواصفات التقدير الجيد

Good Estimation Specifications

يمكن إجمال هذه المواصفات ، كما أوردها **W. Royce (1998)** ، فيما يلي :

① أن يكون مقبولاً من جميع من له علاقة بتطوير المشروع البرمجي **(Stakeholders)**.

② أن تكون مبنياً على موديل لحساب التكلفة **(Cost Estimation Model)** أو أكثر بحيث يكون هذه الموديلات معروفة وموثوقاً بها ومجربة في حساب تكلفة المشاريع البرمجية.

③ أن يكون مبنياً على أساس قاعدة بيانات خاصة بمشاريع سبق إنجازها بحيث تكون مشابهة للمشروع تحت التطوير من حيث : العميل ، وضوابط العمل ، وعمليات وتقنيات التطوير ، وبيئة التطوير ، والمتطلبات ، ومدة الإنجاز والمطورين ، مع الأخذ في الاعتبار المخاطر الناتجة من أن التجارب السابقة لا تمثل دائماً مؤشراً جيداً للنتائج المستقبلية.

④ أن تكون معرّفة بطريقة جيدة وبالتفاصيل المطلوبة لفهم مناطق الخطر الرئيسة التي تواجه عملية التطوير، وكذلك لضمان نجاح المشروع.

⑤ أن تستخدم بعض تقنيات تفكيك المشروع **(Project Decomposition)** إلى أجزاء بسيطة مترابطة لتوليد التقديرات الإجمالية لتكلفة المشروع انطلاقا من تقديرات جزئية يسهل التعامل معها.

وفي الحقيقة تُعد بعض المواصفات السابقة (2 , 3 and 5) خيارات وتوجهات لمحاولة التوصل إلى تقديرات موثوقة للكلفة والجهد اللازمين لإنجاز أي مشروع برمجي، والحالة المثلى هي تطبيق هذه الخيارات والتوجهات مجتمعة ، إذ يمكن لكل منها أن يستخدم كعامل ضبط للآخر. فنماذج التقدير التجريبية

(Empirical Estimation Models) ، والتي ترتكز على الخبرات السابقة متمثلة بالبيانات التاريخية ، يمكن أن تستخدم كأداة مكملة لتقنيات التفكيك.

أسئلة تقويم ذاتي

ما هي أهم مواصفات التقدير الجيد لتكاليف المشاريع؟

10. أفكار مفيدة لتقدير تكلفة المشروعات البرمجية

(Software Project Cost Estimation Tips) :

من خلال ما سبق يمكن سرد بعض الأفكار المفيدة في عملية التقدير فيما يلي :

①خذ الوقت الكافي لتقدير التكلفة المناسبة ؛ حيث إن التقدير المندفع يكون غير دقيق ، واعتبر كل خطوة من خطوات تقدير تكلفة المشاريع الضخمة مشروعاً صغيراً.

②كلما أمكن استخدم بيانات موثقة لمشروعات مماثلة تم تنفيذها من قبل ، حيث إنها تقود إلى تقدير أدق. وإن لم تكن مؤسستك قد قامت بالتوثيق الكامل لبيانات تكلفة المشروعات السابقة ، فأغتنم هذه الفرصة للبداية في ذلك ابتداءَ من أول مشروع قادم أو حالي تقوم بإدارته.

③اعتمد في التقدير على القائمين بالتطوير **(Developers)** الفعلي للمشروع ، فالتقدير المعتمد على الأشخاص غير القائمين بالعمل غالباً ما يكون غير دقيق.

④استخدم على الأقل أداة برمجية لتقدير التكلفة ؛ حيث تتضمن مثل هذه البرمجيات على نماذج رياضية معقدة ولكنها مفيدة في عملية التقدير لأنها مبنية على بيانات تاريخية للعديد من المشروعات التي تم تنفيذها من قبل وتتيح لك ضبط التقدير بسرعة وبدون متاعب ، وكذلك لصعوبة فهم وتطبيق هذه النماذج يدوياً.

⑤كلف العديد من الأشخاص المختلفين لإجراء عملية التقدير ، وكذلك استخدم منهجيات تقدير مختلفة ، على أن يكون منها على الأقل أداة تقدير برمجية مبنية على النماذج الرياضية التي تستخدم البيانات التاريخية لضبط عملية التقدير ، ومن ثم قارن جميع النتائج ، وفي حالة اتفاق معظم هذه التقديرات تكون قد حصلت على التقدير الصحيح بدقة كبيرة ، وأما في حالة عدم توافق هذه النتائج فهذا يعني وجود حيثيات في عملية التقدير لم تؤخذ في الاعتبار ؛ حاول معرفة وفهم هذه الحيثيات وعمل توافق بين النتائج السابقة.

⑥أعد تقدير تكلفة المشروع مرات عديدة طوال دورة حياته ، حيث إن دقة التكلفة المقدرة تزداد كلما تقدم تطور المشروع حتى تصل إلى التكلفة الفعلية الصحيحة مع اكتمال المشروع.

⑦ابتدع طريقة تقدير قياسية **(Standardized Estimation Procedure)** بحيث يقبل بها الآخرون ذو الصلة بعملية التطوير ويستطيعون استخدامها ، حيث إنه بهذه الطريقة سوف ينصب تركيزك فقط وبإنتاجية كبيرة على فهم المدخلات والتي تتمثل في أفق المشروع ومسببات التكلفة **(Cost Drivers)** مثل : حجم المشروع ، وأنواع أخرى من التكلفة. أما المخرجات فسوف تعتمد على دقة الطريقة التي ابتدعتها وكذلك على دقة المدخلات والتي سوف تكون في دائرة اهتمامك.

⑧أعطي بعض الجهد لتحسين عملية تقدير تكلفة المشاريع البرمجية في مؤسستك ، وذلك من خلال متابعة المشاريع الجارية ومقارنة التكلفة الحقيقة بعد انتهاء هذه المشاريع بالتكلفة التي قدرت لها من قبل ، ومعرفة الجيد مما قمت به في عملية توقع المجهود **(Effort)** والجدول الزمني **(Schedule)** ، وما لم تأخذه في الحسبان ، وكيفية تحسين هذه العملية.

وكخلاصة لهذا الموضوع يمكن القول بأنه ليس هناك طريقة سريعة تجعلنا مثمنين جيدين **(Good Cost Estimators)** لتكلفة المشاريع البرمجية ؛ حيث إن عملية التقدير الفعالة في حد ذاتها تُعد نتيجة لعمليات عديدة منها : التعليم والتدريب ، والإدارة الجيدة للمشروع ، واستخدام القياسات والمنهجيات والأدوات المناسبة ، والموارد المتاحة ، والجهد الدؤب ، وبيئة التطوير. ولكن مع ذلك يمكنك تحسين عملية التطوير من خلال :

❶ أعد عملية تقدير تكلفة المشروع عند انتهاء كل مرحلة من المراحل الأساسية في دورة حياته (بعد اكتمال مرحلة تحديد مواصفات المتطلبات ، بعد اكتمال مرحلة التصميم المعماري ، بعد استكمال التصميم التنفيذي المفصل ، بعد استكمال عملية البرمجة ، بعد استكمال عملية التكامل ، بعد الاستلام الابتدائي ، .....).

❷ في نهاية المشروع :

- سجل القيم الحقيقة أو القريبة منها : للحجم **(Size)** ، والجهد **(Effort)** ، والجدول الزمني **(Schedule)** ، والتكلفة **(Cost)** ، وفريق العمل **(Staffing)** ، وسجل ذلك في قاعدة البيانات التاريخية **(Historical Database)** الخاصة بشركتك أو مؤسستك.
- قارن بين التكلفة الفعلية للمشروع والتكلفة التي قدرت له من قبل ومعرفة الجيد مما قمت به في عملية توقع المجهود **(Effort)** والجدول الزمني **(Schedule)** ، وما لم تأخذه في الحسبان ، وكيفية تحسين هذه العملية في المستقبل.

وهنا سوف نلخص بعض النقاط المهمة التي يمكن أن تهتم بها في المشروع القادم إن شاء الله :

1) راجع عملية تطوير المشروع البرمجي الحالي وأجب عن الأسئلة التالية :

- هل تتم بطريقة عشوائية؟ أو تتم طبقاً لقواعد وبنيات تركيبية محددة تقوم بإتباعها؟. وإذا كانت عشوائية حاول التخلص من هذه العشوائيات أو على

الأقل التقليل منها.

2) قم بعمل مسودة أولى من مستند خطوات إجراء عملية التقدير ، واتبع هــذه الخطوات في عملية التقدير وسجل ما هو مناسب وغير مناسب منها ، ومن ثم قم بتحديث هذه المسوّدة.
3) خذ وقتاً كافياً لكي تقدر تكلفة المشروع بطريقة دقيقة.
4) قرر متى يتم إعادة تقدير تكلفة المشروع ، مع وضع المراحــل أو المعــالم الأساسية التي تريد إعادة عملية التقدير بعد اكتمالها على مخطط المشروع.

5) الخطو بعناية واهتمام وبتأني في عملية التحسين يقود لتقصير الطريــق إلــى تقدير جيد ودقيق للمشروع ؛ حيث إن أخذ خطوات صغيرة بعناية هــي الطريقــة المؤكدة لحدوث تغيير دائم.

أسئلة تقويم ذاتي

ما هي أهم النصائح والأفكار المفيدة التي يمكن أن تسترشد بهــا لتقدير تكلفة المشروعات البرمجية؟

الخلاصة

اشتملت هذه الوحدة على

– التقدير الدقيق لتكلفة المشاريع البرمجية والذي يحوي:

* التقدير الأقل من التكلفة الفعلية وهذا يمكن أن يقود إلى : تكوين فريق عمل قليل العدد مقارنة بالجهد المطلوب مما يؤدي إلى إجهاد هذا الفريق وبالتالي إلى عدم الالتزام بالجدول الزمني.

* تحقيق مستوى أقل من أفق ضمان الجودة المطلوب.

* تحديد جدول زمني قصير المدى لإنجاز المشروع.

* التقدير الأعلى من التكلفة الفعلية يمكن أن يؤدي إلى إضافة موارد غير ضرورية وعدم تحكم كاف لأفق المشروع.

– العوامل التي تؤثر على دقة تقدير تكلفة المشاريع البرمجية وهي:

* درجة دقة تقدير حجم وأفق المشروع.

* وقت عملية التطوير.

* دقة تحديد متطلبات المشروع ومقدار ثباتها أثناء عملية التطوير.

* مقدار ثبات ظروف بيئة التطوير .

* دقة ترجمة هذا الحجم إلى جهد بشري ، وخطة زمنية ، ونفقات مالية.

* قدرات وكفاءات وإنتاجية فريق العمل .

* حجم البرمجيات الجاهزة التي يمكن إعادة استخدامها في المشروع الجديد.

* مدى دراية وخبرة مديري المشاريع وطاقم التطوير بالتغير الهائل في التقنيات المستخدمة في المشاريع.

– طرق تقدير حجم البرمجية :

* طريقة عدد سطور شيفرة المصدر وهي الأكثر انتشاراً في تقدير حجم البرمجية لسهولتها.

* طريقة نقاط الوظيفة وهي مقياس تركيبي يعتمد على وظائف البرمجية وليس على نوع لغة التطوير.

* طريقة نقاط الميزة وقد تم تطوير هذه الطريقة لتقدير أو قياس حجم النظم البرمجية التي تعمل في الوقت الحقيقي.

– الإنتاجية وهي مقياس لتقييم كفاءة المبرمجين المشاركين في فريق تطوير المشاريع البرمجية.

– منهجيات تقدير الجهد وفيها يقدر المجهود من خلال المعادلة التالية:

المجهود = حجم العمل/معدل الإنتاجية.

– تقدير التكلفة: هناك ثلاثة متغيرات أساسية تدخل في تقدير التكاليف الكلية لمشروع تطوير برمجية وهي:

* تكلفة العتاد والبرمجيات شاملة الصيانة.

* تكلفة السفر للعاملين والتدريب.

* التكاليف المدفوعة للمطورين.

– تقدير فترة الجدولة الزمنية وعكس القول إن أنواع التقديرات الخاصة بفترة الجدولة الزمنية هي:

* اقصر فترة جدولية زمنية ممكنة ، حيث أنها مضغوطة لأكثر درجة ممكنة.

* فترة الجدولة الزمنية الفعالة.

* فترة الجدولة الظاهرية.

– مواصفات التقدير الجيد : وهو أن يكون مقبولاً من جميع من له علاقة بتطوير المشروع البرمجي.

* وأن يكون مبنياً على موديل لحساب التكلفة.

و أن يكون مبنياً على أساس قاعدة بيانات خاصة بمشاريع سبق انجازها بحيث تكون مشابهة للمشروع تحت التطوير.

وأن تكون معرفة بطريقة جيدة وبالتفاصيل المطلوبة.

وأن تستخدم بعض تقنيات تفكيك المشروع .

– أفكار مفيدة لتقدير تكلفة المشروعات البرمجية:

وأخذ الوقت الكافي لتقدير التكلفة المناسبة – واستخدم بيانات موثقة لمشروعات مماثلة تم تنفيذها من قبل – اعتمد في التقدير على القائمين بالتطوير الفعلي بالمشروع – استخدام على الأقل أداة برمجية لتقدير التكلفة – كلف العديد من الأشخاص المختلفين لإجراء عملية التقدير– اعداد تقرير تكلفة المشروع مرات عديدة – ابتكار طريقة تقدير قياسية – إعطاء بعض الجهد لتحسين عملية تقدير تكلفة المشاريع.

لمحة مسبقة عن الوحدة التالية

نتناول في الوحدة التالية "النماذج الخوارزمية لتقدير البرمجيات" تعريف نماذج التقدير الخوارزمية ، والشكل العام لها ، وبيان الصعوبات التي تعاني منها ، وكيفية قياس مدى دقتها ، وبيان الأسس التي يتم على أساسها تقويمها ، وشرحاً وافياً لأهم أنواعها، بالإضافة إلى شرح تفصيلي للشروط التي يجب أن تؤخذ في الاعتبار عند تقويم أدوات تقدير تكلفة المشاريع البرمجية.

اجابات التدريبات

تدريب (1)

مقارنة بين مقياس SLOC ومقياس FP [12]

مقياس سطور شيفرة المصدر (SLOC)	مقياس عدد نقاط الوظيفة (FP)
معتمد على التناظر (Analog-Based)	معتمد على المواصفات (Specification Based)
معتمد على لغة التطوير (Language Dependent)	غير معتمد على لغة التطوير (Language Independent)
يكيف وفقاً للتصميم Design-Oriented	يكيف طبقاً للمستخدم User-Oriented
يتغير كدالة في اللغات Variations as a Function of Languages	يتغير كدالة في اصطلاحات العد Variations as a Function of Counting Conventions
يمكن تحويله إلى مقياس نقاط الوظيفة Convertible Function Points to	يمكن تحويله إلى مقياس سطور شيفرة المصدر Expandable to Source Line of Code

تدريب (2)

العوامل المؤثرة على إنتاجية المبرمجين

العامل	الوصف
خبرة مجال التطبيق (Application Domain Experience)	الخبرة بمجال التطبيق شيء أساسي لعملية التطوير ، وبالتالي فإن المهندسين الذين لديهم خبرة بمجال التطبيق سيكونون أكثر إنتاجية من الجدد.
جودة العملية (Process Quality)	تلعب جودة عملية التطوير دوراً كبيراً في تقدير الإنتاجية.
حجم المشروع (Project Size)	كلما زاد حجم المشروع كلما زاد الوقت لاتصالات فريق التطوير ، وبالتالي يقل الوقت المأخوذ في التطوير، وبالتالي تقل الإنتاجية الفردية.
التقنية المستخدمة (Used Technology)	التقنية المستخدمة تلعب دور كبير جدا في تحسين الإنتاجية : مثل استخدام أدوات هندسة البرمجيات (CASE Tools) .
بيئة العمل (Working Environment)	توفير بيئة عمل هادئة ، مع تقسميها إلى أقسام لكل منها عمل خاص ، تلعب دوراً كبيراُ في زيادة الإنتاجية.

تدريب(3)

للعديد من الأسباب أهمها :

① أنها عملية دقيقة تتضمن العديد من المتغيرات التي يمكن أن تتدخل في تحديد التكلفة النهائية للنظام والجهد المستثمر في تطويره ، لذا ربما يتم التقدير المبدئي لتطوير البرمجية بناءً على تعريف الحد الأقصى للمتطلبات بدون أخذ المتغيرات الأخرى في الحساب.

② عملية التقدير يجب أن تتم في بداية المشروع ، وهذا يعني أن تقدير التكلفة تتم قبل تصميم المستوى الأعلى للنظام **(Project Top-Level Design)** ، وكذلك قبل التعرف الجيد على الوظائف الأساسية للمشروع ، وهذا في حد ذاته يمثل خطورة كبيرة على عملية التقدير.

③ عدم وجود قاعدة بيانات تاريخية موثوق بها لقياسات عملية تقدير تكلفة المشاريع البرمجية يمكن الاعتماد عليها.

④ ندرة الخبراء المتدربين في مجال تقدير تكلفة المشاريع البرمجية.

⑤ عدم وضوح وعدم فهم العلاقة بين الإنتاجية **(Productivity)** والجودة **(Quality)** : فكل القياسات المعتمدة على الحجم ووحدة الزمن **(Volume / Unit Time)** بدون أخذ الجودة في الاعتبار تُعد غير دقيقة ، حيث إنه يمكن زيادة الإنتاجية بصورة كبيرة على حساب الجودة ، بالإضافة إلى أنه ربما لم يتم تحديد ومعرفة مهارات وإمكانيات العديد من مهندسي فريق العمل الذين سيعملون في المشروع بالدقة المطلوبة.

مسرد المصطلحات

الملفات الداخلية المنطقية ogical Internal Files :

و تشمل جميع الملفات المنطقية الرئيسة ، والتي يمثل كل منها تجميع منطقي لمجموعات من البيانات التي قد تكون جزءاً من قاعدة بيانات واسعة (مثل جدول في قاعدة بيانات علائقية) ، أو جزءاً من ملف مستقل (Single Flat File).

ملفات المواجهة الخارجية External Interface Files :

وتشمل جميع الملفات التي تستخدم لنقل أو لمشاركة البيانات مع نظام آخر.

الإنتاجية Productivity :

مقياس لتقييم كفاءة المبرمجين المشاركين في فريق تطوير المشاريع البرمجية ، ولا بد من تقديرها من قبل مديري المشاريع من خلال قياس بعض سمات البرمجية ومقارنته بالجهد الكلي المطلوب للتطوير ، وهناك نوعين من المقاييس أو السمات المستخدمة.

الجهد Effort :

كمية العمل الإنساني (Human Work Amount) المبذولة لتطوير المشروع ، ويمكن أن يعبر عنها باستخدام العديد من الوحدات ، مثل : شخص ــ ساعات (Person - Hours) ، شخص ــ أيام (Person- day) ، شخص ــ أسابيع (Person - Weeks) ، شخص ــ شهور (Person - Months) ، شخص ــ سنين (Person - Years).

التقدير المبني على التشابه Estimation by Analogy :

تقدير تكلفة المشروع بناءً على تكلفة مشاريع مماثلة سابقة في مجال التطبيق نفسه ، حيث يتم تقدير تكلفة المشروع قيد الدراسة بالتشابه ، وتمتاز بالدقة إذا

توافرت بيانات لمشاريع مماثلة للمشروع تحت التقدير ، ويعيبها أنها مستحيلة الحدوث إذا لم يكن هناك مشاريع مماثلة.

التقدير للفوز Pricing to Win :

تقدير تكلفة المشروع البرمجي بناءً على مقدرة العميل على الإنفاق عليه ، أي أن الجهد يقدر بناءً على ميزانية العميل وليس على أداء البرمجية.

عدد سطور شيفرة المصدر Source Lines-of-Code "SLOC" :

قياس عدد سطور الشيفرة البرمجية بعد استبعاد سطور التعليقات **(Comments Lines)** ، والسطور الخالية **(Blank Lines)**.

نمذجة التكلفة الخوارزمية Algorithmic Cost Modeling :

وهي منهجية تُستخدم لتقدير الجهد والتكاليف تعتمد على المعادلات (Formulaic Approach)

رأي الخبير Expert Judgment :

وهي منهجية تُستخدم لتقدير الجهد والتكاليف وفيها يتم الاستفادة من الخبراء المتخصصين في مجال تطوير المشاريع البرمجية حيث يقوم كل منهم بعمل تقدير لتكلفة المشروع حيث يتم مناقشتها والوصول إلى رأي نهائي متفق عليه لهذه التكاليف ، وتعتمد دقة تقدير هذه التكلفة على مدى دراية هؤلاء الخبراء بمشاريع مماثلة.

التقدير المبني على التشابه (Estimation by Analogy) :

وهي منهجية تُستخدم لتقدير الجهد والتكاليف وفيه يتم تقدير تكلفة المشروع بناءً على تكلفة مشاريع مماثلة سابقة في مجال التطبيق نفسه ، حيث يتم تقدير تكلفة المشروع قيد الدراسة بالتشابه ، وتمتاز بالدقة إذا توافرت بيانات لمشاريع مماثلة

للمشروع تحت التقدير ، ويعيبها أنها مستحيلة الحدوث إذا لم يكن هناك مشـــاريع مماثلة.

التقدير للفوز Pricing to Win :

وهي منهجية تُستخدم لتقدير الجهد والتكاليف وفيه يتم تقدير تكلفة المشروع البرمجي بناءً على مقدرة العميل على الإنفاق عليه ، أي أن الجهد يقدر بناءً على ميزانية العميل وليس على أداء البرمجية.

"أقصر فترة جدولة زمنية ممكنة" Shortest Possible Schedule:

هي أقصر فترة جدولة زمنية ممكنة والتي لا يمكن تعديها لأي مشـــروع وبـــأي وسيلة من الوسائل حيث لا يمكن ضغط فترة الجدولة الزمنية لأصغر من ذلـــك ، حيث إنها مضغوطة لأكثر درجة ممكنة.، ويجب عليك معرفتها فقط بدون محاولة تجاوزها.

فترة الجدولة الزمنية الفعالة Efficient Schedules :

أقصر فترة جدولة زمنية يمكن الوصول إليها بدون التفكير في الوصول إلى "Shortest Possible Schedule"

المصطلح بالإنجليزية	معناه بالعربية
4GLs	**لغات الجيل الرابع**
Analog-Based	**معتمد على التناظر**
AVC	**المتوسط الحسابي لعدد السطور للغة معينة**
Backup	**نسخ احتياطي**
Blank Lines	**السطور الخالية**
CASE Tools and Programs Generators	**هندسة البرمجيات بمساعدة الكمبيوتر ومولدات البرامج**
Client-server Applications	**تطبيقات خادم العميل**
Comments Lines	**سطور التعليقات**
Commercial Cost Models	**نماذج تقدير التكلفة التجارية**
Components-Based Software Engineering	**هندسة البرمجيات المبنية على المكونات**
Control Signals	**إشارات التحكم**
Conversion	**التحويل**
Convertible Function Points to	**يمكن تحويله إلى مقياس نقاط الوظيفة**
Cost	**التكلفة**
Crucial	**حرجه**
Design-Oriented	**وفقاً للتصميم**
Effort	**الجهد**
Expandable to Source	**يمكن تحويله إلى مقياس سطور شيفرة المصدر**
External Interface Files	**ملفات المواجهة الخارجية**
Forms	**النماذج**
Feature Points	**نقاط الميزة**
Function Points "FP"	**نقاط الوظيفة**
Function-oriented	**المبني على الدوال**
Hard Line Break	**بداية سطر جديد**

المصطلح بالإنجليزية	معناه بالعربية
Historical data analysis	**التحليل التاريخي للبيانات**
Immediate Input	**دخل فوري**
Immediate Outputs	**خرج فوري**
Influence Multiplier	**مضروب التأثير**
Initial Product Definition Stage)	**مرحلة التعريف الأولى للمنتج**
Inquiries	**الاستفسارات**
Installation	**التثبيت**
Iterative Steps	**الخطوات التكرارية**
Language Dependent	**معتمد على لغة التطوير**
Language Independent	**غير معتمد على لغة التطوير**
Logical Internal File	**الملفات الداخلية المنطقية**
Logical Lines	**عدد السطور المنطقية**
Mainframe System –Based Applications	**التطبيقات المبنية على نظم الحاسبات المركزية**
Management Information Systems "MISs"	**نظم إدارة المعلومات**
Methods of Software Size Estimation	**طرق تقدير حجم البرمجية**
Object-oriented	**البرمجة الكينونية**
Off Shelf Components	**المكونات المتناولة**
OOPL	**اللغات الكائنية المنحى**
Optimistic Estimates	**التقديرات المتفائلة**
Over-Estimating Cost	**التقدير الأعلى من التكلفة الفعلية**
Pessimistic Estimates	**والتقديرات المتشائمة**
Physical Format	**الشكل الفيزيائي**
Physical Lines	**عدد السطور الفيزيائية**
Project Key Milestones	**المعالم الأساسية للمشروع**

المصطلح بالإنجليزية	معناه بالعربية
Recovery	**استعاده دوريه يمكن الوثوق بها**
Requirements Specification	**مرحلة إعداد مواصفات المنتج**
Resources	**الموارد**
Schedules	**الجداول الزمنية**
Semicolons	**الفصلات المنقوطة**
Sets of Open-Closed Braces	**مجموعات أزواج الأقواس**
Single Flat File	**ملف مستقل**
Software Functionality	**وظائفية البرمجية**
Software Instructions	**عدد التعليمات البرمجية**
Software Projects Cost Estimation	**تقدير تكلفة المشروعات البرمجية**
Source Lines-of-Code "SLOC"	**عدد سطور شيفرة المصدر**
Specification Based	**معتمد على المواصفات**
Staffing	**فريق التطوير**
Synthetic Measure	**مقياس تركيبي**
Total Cost	**التكاليف الكلية**
UFC (Unadjusted Function-Point Count)	**ونقاط الدالة غير المعدلة**
Under-Estimating Cost	**التقدير الأقل من التكلفة الفعلية**
User-Oriented	**طبقاً للمستخدم**
Variations as a Function of Counting Conventions	**يتغير كدالة في اصطلاحات العد**
Variations as a Function of Languages	**يتغير كدالة في اللغات**
Web-Based Applications	**التطبيقات المعتمدة على الويب**
Weight	**وزن**

المراجع

أولاً : المراجع العربية

[1] روجر بريسمان ، "هندسة البرمجيات" ، ترجمة مركز التعريب والترجمة بالدار العربية للعلوم ، الطبعة الأولى ، 2004م.

[2] مهندس عبد الحميد بسيوني ، "أساسيات هندسة البرمجيات" ، دار الكتب العلمية للنشر والتوزيع ، القاهرة ، 2005 م.

ثانياً : المراجع الإنجليزية :

[3] Ian Somerville , "Software Engineering", Addison Wesley, 2001.

[4] Ronald J. Leach, "Introduction to Software Engineering", CRC Press, 1999.

[5] Douglas Bell , "Software Engineering : A Programming Approach", 3rd Edition, Addison Wesley.

[6] Shari Pfleeger, "Software Engineering - Theory and Practice", 2nd Edition.

[7] B. W. Boehm, Software Engineering Economics, Englewood Cliffs, NJ: Prentice-Hall, 1981.

[8] Steven C. McConnel , "Estimation", Chapter 8, Rapid Development, Microsoft Press, 1996 , www.construx.com/stevemcc

[9] Boehm, et al , "Cost Models for Future Life Cycle Processes: COCOMO 2.0", 1995.

[10] Karen Lum and et. al , "Handbook for Software Cost Estimation" , Jet Propulsion Laboratory Pasadena, California.

WWW.ceh.nasa.gov/downloadfiles/ Web%20Links/cost_hb_public-6-5.pdf

[11] Reifer, D. , Boehm, B., and Chulani, S., "The Rosetia Stone: Making COCOMO81 Estimatea Work with COCOMOII, "Crosstalk : The Journal of Defense Software Engineering", February 1999.

[12] Albrecht, A. J., "Measuring Application Development Productivity", Proc. IBM Application Development Symposium. Monterey, CA, October 1979.

[13] Jones, Capers. Applied Software Measurement: Assuring Productivity and Quality, New York: McGraw-Hill, 1991.

[14] Albrecht, A. J., and J. E. Gaffiney, " Software Function, Source Line of Code and Development Effort Prediction : A Software Science Validation", IEEE Trans. Software Engineering, November 1983.

[15] U.S. Air Force's Software Technology Support Center , " Chapter 13 Software Estimation, Measurement, and Metrics"

http://www.stsc.hill.af.mil/resources/tech_docs/gsam3/chap13.pdf

[16] Conte, S. D., Dunsmore, H. E., and Shen, V. Y., Software Engineering Metrics and Models, Benjamin/Cummings Publishing Co., Inc., Menlo Park, CA 1986.

[17] Guillermo Sebastian Donatti , "Software Development Effort Estimations Through Neural Networks" , Faculty of Mathematics , Astronomy and Physics , Cordoba National University ,Cordoba, May 2005.
http://www.freewebtown.com/sdeetnn/web/docs/sdeetnn.pdf

[18] Putnam, L., and W. Myers, "Measure for Excellence ", Yourdon Press, 1992.

[19] "Modern Empirical Cost and Schedule Estimation Tools",

عنوان مقال منشور على الموقع :

http://www.dacs.dtic.mil/techs/estimation/comparison.shtml

[20] Martin Glinz Arun Mukhija , "COCOMO (Constructive Cost Model)" , Seminar on Software Cost Estimation , WS 2002 / 2003 , Presented by Nancy Merlo – Schett , Requirements Engineering Research Group , Department of Computer Science , University of Zurich, Switzerland.

[21] Danfeng Hong. "Software Cost Estimation" , Department of Computer Science , University of Calgary , Alberta, Canada T2N 1N4.
http://kdataserv.fis.fc.ul.pt/~aribeiro/cost/swCostEstimation.html

[22] Chapter 4 : Quality Development Costs and Schedule .

http://www.ii.metu.edu.tr/~ion545/demo/section1.3.html

[23] Kim Johnson , "Software Cost Estimation: Metrics and Models" , Department of Computer Science , University of Calgary , Alberta , CANADA T2N 1N4.
http://sern.ucalgary.ca/courses/seng/621/W98/johnsonk/cost.htm

[24] Boehm, B.W., Abts, C., Clark, B., and Devna Ni-Chulani. S. (1997). COCOMO II Model Definition Manual. The University of Southern California.

[25] COCOMO II Model Definition Manual , Version 2.1 , 1995-2000

Center For Software Engineering , USC.

[26] Hareton Leung and Zhang Fan , "Software Cost Estimation" , Dept. of Computing , Hong Kong Polytechnic University.

[27] Kathleen Peters, "Software Project Estimation", Software Productivity Center Inc.,

http://www.spc.ca/downloads/resources/estimate/estbasics.pdf

ثالثاً : مواقع على شبكة الإنترنت تم الاستفادة منها :

[28] http://sunset.usc.edu/publications/TECHRPTS/2000/usccse2000-505/usccse2000-505.pdf ,
"Software Development Cost Estimation Approaches - A Survey"

[29] www.comp.lancs.ac.uk/computing/resources/IanS/SE7/SampleChapters/ch26.pdf ,
" Software cost estimation"

[30] www.classes.cecs.ucf.edu/eel6883/ berrios/slides2/CH7-art-3-4-5.pp ,
" Software Cost Estimation "

- **COCOMO II web page can be found at : sunset.usc.edu/research/COCOMOII/index.html**
- **USC COCOMO II.1999.0 Software (Implementations of Post-architecture and Early Design models) : http://sunset.usc.edu/research/COCOMOII/index.html**


الوحدة السابعة
النماذج الخوارزمية
لتقدير البرمجيات

محتويات الوحدة

مقدمة

تمهيد

عزيزي الدارس ،،

أهلاً بك في الوحدة السابعة من مقرر " هندسة البرمجيات (1)" ، والتي تتناول النماذج الخوارزمية لتقدير البرمجيات في أربعة أقسام رئيسة :

-القسم الأول نتناول فيه مقياس الصناعة لدقة تنبأ نموذج التكلفة : نتناول فيه عدد من مقاييس دقة النموذج التجريبي.

- القسم الثاني من الوحدة يتناول عدداً من المعايير لتقويم نماذج تقدير التكلفة.

- القسم الثالث من الوحدة يبحث في أنواع النماذج الخوارزمية ، ونتناول عدداً من النماذج مع توضيح مميزات كل نموذج وعيوبه.

- في القسم الرابع نتناول أدوات تقدير تكلفة المشروعات البرمجية ونتعرف هنا على العديد من السمات والشروط التي يجب أن تؤخذ بعين الاعتبار عند تقويم أدوات تقدير تكلفة المشاريع البرمجية.

تجد في متن هذه الوحدة تدريبات وأسئلة تقويم ذاتي وأمثلة مع حلولها ، أرجو أن تفيد منها .

وختاماً نرحب بك عزيزي الدارس في هذه الوحدة مرة أخرى، ونرجو أن تكون وحدة مفيدة لك، وأن تساهم معنا في تقدمها وتقييمها.

أهداف الوحدة

عزيزي الدارس، بنهاية دراسة هذه الوحدة ينبغي أن تكون قادراً على أن :

- تصف النماذج الخوارزمية المستخدمة في تقدير تكلفة المشـــاريع البرمجية.
- تُعدد أهم الصفات المشتركة بين النماذج الخوارزمية المختلفة.
- تقيس مدى دقة تقدير تكلفة المشـــاريع البرمجيـــة لأي نمـــوذج خوارزمي.
- تقيم نماذج تقدير تكلفة المشاريع البرمجية من خلال استخدام أهم المعايير الخاصة بذلك.
- تعدد أهم أنواع النماذج الخوارزمية المستخدمة في تقـــدير تكلفـــة المشاريع البرمجية.
- تستخدم نموذج بوتنام لتقدير تكلفة المشاريع البرمجية.
- تستخدم نموذج بوهم (كوكومو) بأنواعه المختلفة في تقدير تكلفـــة المشاريع البرمجية.

توطئة

نماذج التقدير الخوارزمية **(Algorithmic Estimation Models)** هي : عبارة عن دوال رياضية مستنبطة على أساس نظري **(Theory–Based Models)** (مثل : **Putnam SLIM Model)** ، أو على أساس تجريبي **(Empirical-Based Model)** باستخدام التحليل الانحداري **(Regression Analysis)** على بيانات تاريخية **(Historical Data)** مجمعة من مشاريع برمجية سابقة (مثل : **COCOMO Models**) ، هذا بالإضافة إلى البديهة **(Intuition)** والخبرات **(Experiences)**، أي أن معظم هذه النماذج في النهاية تعتمد على مجموعة من البيانات التاريخية المأخوذة من عينات مشاريع محددة ، لذلك يجب معايرتها قبل استخدامها لتلائم مع بيئة التطوير التي سوف تستخدم بها والتعامل مع نتائجها بحذر ، حيث إنه لا يوجد نموذج ملائم لشتى أنواع البرمجيات وجميع بيئات التطوير البرمجي.

والشكل العام لمعظم هذه النماذج يأخذ الصيغة الشائعة الموضحة بالمعادلة رقم (1) :

$$E = A^* (Size)^B * EMF \text{.....(1)}$$

حيث "**E**" تمثل الجهد المقدر بشخص ــ شهور **(Effort in PM)** ، و"**A**" ، و"**B**" فهي ثوابت تُشتق تجريبياً ، حيث الثابت "**A**" يعتمد على هيكلية وخبرة المؤسسة أو الشركة المطورة **(Organization Dependent)** ، وكذلك على نوع المشروع الذي تحت التطوير ، و **الثابت "B"** يعكس تأثير بعض العوامل المتعلقة بأسلوب التطوير مثل : مرونة التطوير **(Development Flexibility)** ، ونضج عمليات التطوير **(Development Process Maturity)** ، ومدى تناغم وتفاهم فريق التطوير **(Team Cohesion)** ، **ومنهجيات إدارة المخاطر (Risk Management Techniques)** ، ويتناسب طردياً مع زيادة حجم المشروع ، ويعكس الحقيقة التي تقول إن التكلفة تزداد بزيادة الحجم ولكن بطريقة غير خطية ، حيث إنه بزيادة

حجم المشروع يزداد حجم فريق التطوير وتقل الإنتاجية نتيجة الفاقد الناتج من كثرة الاتصالات بين الفريق ، وزيادة المجهود الذي يبذل في إدارة الفريق ، أما **"Size"** فيمثل حجم البرمجية ، و**"EMF"** ثابت يمثل عامل ضبط الجهد **(Effort Adjustment Factor)** لضبط الجهد ليتضمن تأثير موجهات التكلفة **(Cost Drivers)** .

ويمكن استخدام النماذج الخوارزمية في تقدير تكلفة المشاريع البرمجية كأداة أساسية أو كأداة ثانوية للتحقق من النتائج بناءً على حالة المشروع ، فمثلاً في حالة نضج متطلبات المشروع وفهم مواصفات المتطلبات والتصميم فهماً جيداً يفضل استخدام منهجية التناظر في عملية التقدير كمنهجية أساسية واستخدام منهجية النماذج الخوارزمية كمنهجية ثانوية للتحقق من التقدير ، أما في حالة تقدير تكلفة المشروع في بداية دورة حياته مع متطلبات غير واضحة فيمكن استخدام النماذج الخوارزمية كأداة أساسية بجانب التناظر مع مشاريع مماثلة لمعايرة هذه النماذج. وأيضاً يمكن استخدام النماذج الخوارزمية للتفاضل بين مجموعة من منهجيات التطوير من خلال تحليل تأثيراتها النسبية على عملية التطوير.

وإجمالاً تتصف معظم هذه النماذج بما يلي :

- استخدام معادلات رياضية لإجراء عملية التقدير.
- المعادلات الرياضية معتمدة على النظريات الرياضية أو البيانات التاريخية.
- المدخلات المستخدمة **(Used Inputs)** في عمليات التقدير مثل : عدد سطور الشيفرة **(LOC)** ، عدد نقاط الدالة **(FP)** ، مسببات أخرى للتكلفة **(Other Cost Drivers)**.
- دقة هذه النماذج يمكن أن تُحسن عن طريق معايرتها في نفس بيئة التطوير التي سوف تستخدمها في عملية التقدير.

ولسوء الحظ تعاني معظم هذه النماذج من الصعوبات الأساسية التالية :

- غالباً ما يكون من الصعب تقدير حجم البرمجية في مرحلة متقدمة من دورة حياة المنتج البرمجي ، وذلك لأن حجم البرمجية يعتمد على قرارات التصميم **(Design Decisions)** والتي غالباً لا تحدد في بداية دورة حياة المنتج البرمجي ، حيث تشمل هذه القرارات مثلاً : لغة التطوير والتي تؤثر تأثيراً مباشراً في تحديد حجم البرمجية ، نوع قاعدة البيانات التي سوف تستخدم ، كمية الشيفرة التي سوف يعاد استخدامها ، وخلاف ذلك. ورغم أنه في هذه الحالة يمكن اللجوء إلى نقاط الدالة أو نقاط الكائن كمقياس للحجم بدلاً من عدد سطور الشيفرة ، إلا أنها غالباً ما تكون غير دقيقة.
- تقدير تأثير العوامل التي تساهم في تحديد قيم الثوابت **"B"** ، **"EMF"** غير موضوعية **(Subjective)** ، حيث إنه لا يوجد تعريفات ملائمة لهذه العوامل مما يجعلها شبه مبهمة ، وبالتالي يعتمد تقديرها على الشخص المكلف بتحديدها وعلى مدى خبرته بنوعية المنتج تحت التطوير ، لذا فهذه الثوابت يمكن أن تتغير من شخص إلى آخر وفي بيئة التطوير نفسها وللمنتج نفسه ، وبالتالي تختلف نتيجة التقديرات من شخص إلى آخر.
- معظم هذه النماذج تعاني من عدم دقة تمثيل تأثير الإنتاجية **(Productivity)** على عملية التقدير ، وخصوصاً عند استخدام لغات الجيل الرابع ، وبالتالي فإن قيم التقديرات تكون محل شك ، ويجب إجراء دراسات إضافية في هذا المجال.

أسئلة تقويم ذاتي

ما هو الشكل العام لمعظم نماذج التقدير الخوارزمية– عرف مفردات هذا الشكل العام؟.

ما هي أهم السمات العامة التي تتصف بها نماذج التقدير الخوارزمية؟.

1. مقياس الصناعة لدقة تنبؤ نموذج التكلفة

Industry Measure for Cost Model's Predictive Accuracy :

يمكن قياس مدى دقة النموذج التجريبي في تقدير المشاريع البرمجيـــة مـــن خلال حساب قيمة الخطـــأ النســـبي **(Magnitude of Relative Error "MRE")** باستخدام المعادلة التالية :

$$\mathbf{MRE} = \left(\frac{\left(\left| E_{pred} - E_{act} \right| \right)}{E_{act}} \right)$$

أو يمكن من خلال حساب متوسط قيمة الخطأ النسبي **(Mean Magnitude of Relative Error "MMRE")** الأكثر دقة باستخدام المعادلة التالية :

$$\mathbf{MMRE} = \frac{1}{n} \cdot \sum_{i=1}^{i=n} \left(\frac{\left(\left| E_{pred} - E_{act} \right| \right)}{E_{act}} \right)_i$$

حيث **"n"** تمثل عدد عينة المشاريع/التقـــديرات تحـــت الاختبـــار **(Number of Projects/Estimates Under Test)** ، و $\mathbf{E_{pred}}$ قيمة الجهد التي تنبأ بها النمـــوذج **(Predicted Value)** ، و $\mathbf{E_{act}}$ هي القيمة الحقيقة **(Actual Value)** للجهد، ويجب ألا تزيد قيمة **"MMRE"** عن 25% في حالة كون نتائج النموذج مرضية.

هذا بالإضافة إلى وجود العديد من مقاييس اختبار الدقة في صناعة تطـــوير البرمجيات لتقويم أداء نماذج التقدير التجريبية لتكلفة المشاريع البرمجية ، أحد هـــذه المقاييس ينظر لقدرة النموذج على التنبؤ بالجهد على مستوي عينة مـــن التطبيقـــات غير المترابطة عند مستوى معين من الدقة **(L)** ، ويُعرف بالمعادلة رقم (2) :

$$\mathbf{PRED\ (L) = K / N \ldots\ldots\ldots (2)}$$

حيث **"K"** يمثل عدد مجموعة **(Subset)** من عينة المشـــاريع /التقـــديرات تحـــت الاختبار ذات الدقة الواقعة ضمن المدى المقبول من قيمة الخطأ النسبي **(L=MRE)**

و"N" عدد عينة المشاريع/التقديرات تحت الاختبار ، "L" مستوى دقة التنبؤ ويمثل متوسط قيمة الخطأ النسبي **(MRE)** ، ويجب ألا يقل هذا المقياس عن 75%. لذا فقد اقترحت بعض الدراسات التجريبية **[16]** استخدام هذا المقياس للدلالة على دقة النموذج عن قيمة خطأ نسبي يساوي 0.25 **(L=MRE=0.25)** ، ويجب ألا يقل **PRED(0.25)** عن 0.75.

2. تقويم نماذج تقدير التكلفة

Evaluating of Cost Estimation Models :

تستخدم المعايير التالية لتقييم نماذج تقدير تكلفة المشاريع البرمجية [7] :

①**التعريف (Definition) :** هل عرف النموذج أنواع التكلفة المقدرة وتلك المستبعدة من عملية التقدير؟.

مثال (1) : **تم تقدير الجهد لعدد (100) مشروع باستخدام نموذج تجريبي معين ، فكانت النتائج كتالي :**

1. **عدد (25) تقدير من المائة بعيد عن التقدير الحقيقي (مقدار خطأ 200%).**
2. **عدد (15) تقدير من المائة يساوي التقدير الحقيقي (مقدار الخطأ 0%).**
3. **عدد (60) تقدير من المائة يساوي 70 % من التقدير الحقيقي (مقدار الخطأ 30%).**

أحسب PRED(0.25) ، PRED(0.3)، وفسر النتائج التي سوف تحصل عليها.

الحل :

- **PRED(0.25) = 0.15 ، ومعنى ذلك أن 15 تقدير من الـ 100 مقبولة في حالة اختيار الحد الأعلى لقيمة الخطأ النسبي تساوي 0.25 ، بمعنى أن احتمالية تقدير مشروع باستخدام هذا النموذج بنسبة خطأ نسبي يساوي أو يقل عن 25% تساوي 15%.**
- **PRED(0.30) = 0.75 ، ومعنى ذلك أن 75 تقدير من الـ 100 مقبولة في حالة اختيار الحد الأعلى لقيمة الخطأ النسبي تساوي 0.30 ، بمعنى أن احتمالية تقدير مشروع باستخدام هذا النموذج بنسبة خطأ نسبي يساوي أو يقل عن 30% تساوي 75%.**

②**الدقة (Fidelity) :** هل التكاليف المقدرة قريبة من التكاليف الحقيقية ، وما نسبة هذا التقارب؟.

③**الموضوعية (Objectivity) :** هل يتجنب النموذج تخصيص معظم متغيرات تكاليف البرمجية (مثل التعقيد **"Complexity"**) لعوامل غير موضوعية **(Subjective)** تؤدي إلى معايرة النموذج على نحو ردئ؟ وهل من الصعب ضبط هذه العوامل للحصول على النتائج التي يريدها المستخدم؟.

④**البناء الاستدلالي (Constructiveness)** : هل يمكن للمستخدم أن يفسر الأسباب التي أدت إلى نتائج التقدير الناتجة من استخدام النموذج؟ . وهل يساعد النموذج المستخدم على فهم الخطوات التي يجب أن تتم للحصول على نتائج جيدة؟.

⑤**التفاصيل (Details)** : هل النموذج ملائم لتقدير نظم برمجية تتكون من العديد من الوحدات والنظم الفرعية ؟ وما مدى هذه الملائمة من حيث سهوله تكيفه مع هذه النظم وإعطاء تقسيم دقيق للنشاطات والمراحل المختلفة الخاصة بتطوير البرمجية؟.

⑥**الاستقرار (Stability) :** هل التغيرات الصغيرة في العوامل الداخلة في عملية تقدير التكلفة تؤدي إلى تغييرات صغيرة في نتائج تقدير التكلفة؟.

⑦**الأفق (Scope)** : هل يمكن للنموذج أن يستخدم في تقدير تكلفة معظم أصناف المشاريع البرمجية التي يحتاجها المستخدم ؟.

⑧**سهولة الاستخدام (Ease of Use)** : هل من السهل فهم وتعيين مدخلات واختيارات النموذج والتعامل معها بيسر؟.

⑨**التوقع المستقبلي (Prospectiveness)** :هل يتجنب النموذج استخدام المعلومات التي لا يمكن معرفتها بصورة جيدة إلا مع اكتمال المشروع؟.

⑩**التدبير (Parsimony)** : هل يتجنب النموذج استخدام عوامل زائدة عن الحاجة ، أو تلك التي لا تؤثر تأثيراً فاعلاً يمكن إدراكه في نتائج التقدير؟.

3. أنواع النماذج الخوارزمية

Types of Algorithmic Models :

وقد تم اقتراح العديد من النماذج الخوارزمية **(Algorithmic Models)** لتقدير الجهد وفترة الجدولة الزمنية وتكلفة المشاريع البرمجية ، منها القليل ممّا هو مستنتج على أساس نظري **(Theory–Based Models)** ، أما الكثير منها فهو مستنتج على أساس تجريبي **(Empirical-Based Models)** وهي جمعياً متشابهه من ناحية الشكل والمفهوم ، ولكنهم يستخدمون قيم مختلفة للعوامل **(Parameters)** الداخلة في تكوينها ، وسوف نتناول هنا أهم نموذجين وهما : نموذج **Putnam (SLIM Model)** ونموذج **Boehm (COCOMO Model)**.

1.3 نموذج بوتنام (نموذج دورة حياة البرمجيات) [1 , 17] :

The Putnam Model (Software LIfe Cycle Model " SLIM ")

نموذج بوتنام **"SLIM"** تم تطويره في أواخر السبعينات، وهو مبني على أساس أن توزيع عدد فريق العمل في أي وقت خلال تطوير المشروع **(Overall Staff Distribution over Development Time)** يمكن أن يوصف بتوزيع بدالة **Rayleigh** ، والتي تأخذ الشكل العام طبقاً للمعادلة رقم (3) :

$$M(t) = 2\,K\,a\,t\,e^{-at^2} \quad \ldots\ldots\ldots\ldots\ldots\ldots(3)$$

حيث إن :

M(t) : عدد الأشخاص (فريق العمل) العاملين بالمشروع في أي وقت **"t"**.

K : الجهد الكلي اللازم لتطوير المشروع ، وهو يساوي المساحة الكلية تحت المنحنى المبين بالشكل رقم*7.1* ، والذي يمثل الشكل العام لتوزيع دالة **Rayleigh**.

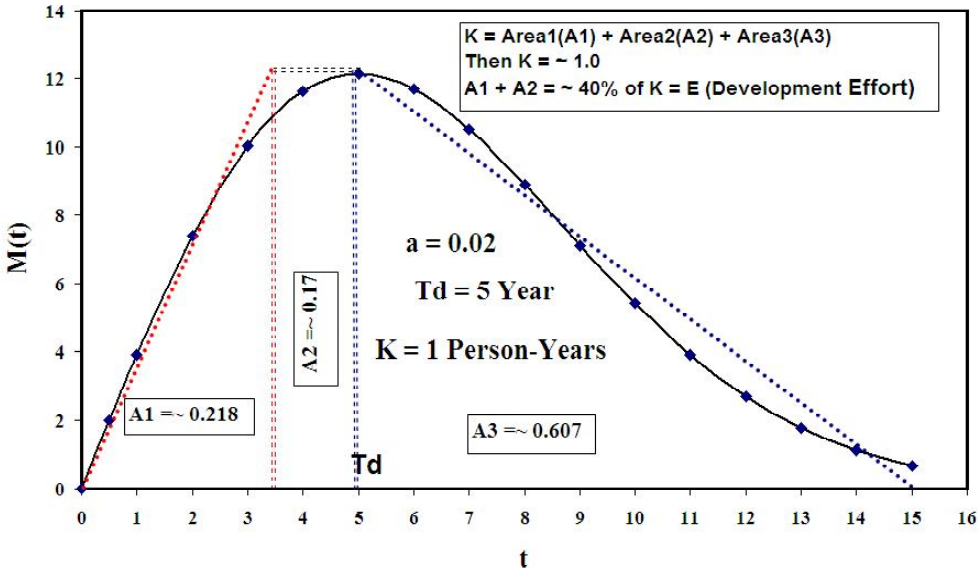
a : يمثل عامل التعجيل **(Acceleration Factor)** ويساوي $(1/2T_d^2)$ ، حيث T_d هو الوقت اللازم لإنهاء عملية تطوير المشروع ، والذي عنده يكون عدد الأشخاص العاملين بالمشروع أكبر ما يمكن **(Peak Staff)** **(dM(t)/dt=0)** ، حيث إن :

$$\text{Peak Staff} = M(T_d) = K/T_d\, e^{-0.5} = (E/0.4T_d) * 0.607$$
$$= \sim 1.5\ E/T_d = \sim 1.5 \text{ Average Staff}$$

حيث إن **"E"** هو الجهد اللازم لتطوير المشروع البرمجي ، وهذا ناتج من أن العلاقة بين الجهد والوقت ليست علاقة خطية.

ومن هنا افترض **Putnam** أن العدد الكلي للأشخاص العاملين في المشروع أو العاملين في كل مرحلة من مراحل تطوير المنتج البرمجي يمكن أن توصف بالمعادلة رقم (4) :

$$M(t) = (K\, t\, e^{-t^2/2T_d^2}) / T_d^2 \quad \text{.....(4)}$$



شكل 7.1 : الشكل العام لتوزيع دالة Rayleigh

وبداية من هذه المعادلة وربطها مع معادلة الإنتاجية : P = S/E ، حيث "P" تمثل الإنتاجية (Productivity) ، و "S" تمثل حجم المنتج البرمجي ، "E" تمثل الجهد المطلوب للتطوير استطاع Putnam أن يربط حجم المنتج البرمجي (S) بوقت التطوير (T_d) ، والجهد الكلي (K) (Person-Year) من خلال المعادلة رقم (5) أو المعادلة رقم (6).

$$S = C\, K^{1/3}\, T_d^{4/3} \quad \ldots\ldots\ldots\ldots\ldots\ldots (5)$$

$$\text{OR} \qquad K = (S / C)^3 * 1/ T_d^4 \ldots\ldots\ldots\ldots\ldots (6)$$

حيث إن "K" هو الجهد الكلي مقيس بشخص ــ سنة (Person-Years) ، و"T_d" الوقت اللازم لإتمام عملية التطوير مقيس بالسنين (Years) ، و "C" ثابت يعكس مدى تأثير :

- مستوى المهارات والخبرات الشخصية لأعضاء فريق العمل.
- حالة البيئة البرمجية بصفة عامة من حيث اللغات وطرق التصميم المستخدمة.
- تعقيدات البرمجية.
- استخدام الأدوات الجيدة لهندسة البرمجيات.
- شمولية نضج عملية البرمجة والممارسات الإدارية.

ويمكن حساب الثابت "C" من خلال المعادلة رقم (7) :

$$C= S / (K^{1/3} * T_d^{4/3}) \quad \ldots\ldots\ldots\ldots\ldots\ldots (7)$$

حيث إنه يمكن حساب "K" و "T_d" من خلال البيانات التاريخية لمشاريع مماثلة بحجم "S" ، ومن ثم استخدام هذه القيمة في تقديرات مستقبلية حتى ظروف بيئة التطوير نفسها.

تلاحظ من المعادلة رقم (6) أن الجهد يتناسب تناسباً عكسياً مع وقت التطور مرفوع إلى الأس الرابع (T_d^4) ، وهذا يعني أنه يمكن خفض الجهد وبالتالي التكاليف إلى النصف بزيادة وقت تطوير المشروع 19% فقط. وقد اقترح Putnam

أن يأخذ الثابت "C" القيم الموضحة بالجدول رقم 7.1 طبقاً لمدى درجة توافر العوامل السابقة التي تعكس مدى تأثيره.

جدول 7.1 : قيم الثابت "C" لنموذج Putnam [17]

قيمة الثابت "C"	الحالة
2000	**في حالة التخطيط الردئ (Poor Rated Program) : لا يوجد منهجية معينة لعملية التطوير من حيث التشفير والتوثيق والمراجعة والإدارة واستخدام الأدوات البرمجية..... وخلافه.**
8000	**في حالة التخطيط الجيد (Good Rated Program) : حيث يوجد منهجية معتمدة لعملية التطوير من حيث التشفير والتوثيق والمراجعة والإدارة ، ولكن استخدام الأدوات البرمجية ليس بالصورة المطلوبة ، وكذلك مستوى المهارات والخبرات الشخصية لأعضاء فريق التطوير متوسطة.**
11000	**في حالة التخطيط الممتاز (Excellent Rated Program) : حيث يوجد منهجية موثقة لعملية التطوير من حيث التشفير والتوثيق والمراجعة والإدارة ، واستخدام الأدوات والمنهجيات البرمجية لأتمتة معظم أعمال عملية التطوير بصورة مهارية ، وكذلك مستوى المهارات والخبرات الشخصية لفريق التطوير عالي.**

نموذج **Putnam** المعتمد على عامل الإنتاجية **(P)** :

قام **Putnam [16]** بعد ذلك بتحليل العديد من البيانات التاريخية لأكثر من 4000 مشروع برمجي ، واستناداً على هذا التحليل قام بتعديل الثابت "C" ليشمل مفهومه عن الإنتاجية **"P"** ، ومن ثم تم الحصول على معادلة لحساب جهد التطوير ، وتُعرف بمعادلة البرمجيات **(Software Equation)** وتأخذ الشكل الموضح بالمعادلة رقم (8) :

$$E = B * (S / P)^3 * 1/ T_d^4 \ldots\ldots\ldots\ldots\ldots (8)$$

حيث إن :

"E" :هو الجهد اللازم لتطوير البرمجية مقدراً بشخص ــ شهور أو بشخص ــ سنين.

"Td": مدة تطوير المشروع مقدرة بالشهور أو بالسنين.

"B" : عامل الخبرات الخاصة ويعتمد على حجم المشروع ، و يمكن تحديد قيمته من الجدول رقم 7.2 حسب حجم البرمجية.

"P" : عامل الإنتاجية ، ويعتمد تأثيره على نفس العوامل التي يعتمد عليها تأثير الثابت "C" في المعادلة رقم (6) ، ويمكن تحديد قيمته من الجدول رقم 7.3.

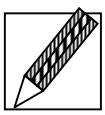
وكما هو واضح من المعادلة رقم (8) أن نموذج **Putnam** له عاملان مستقلان هما : حجم البرمجية **"S"** ويقدر بعدد أسطر الشيفرة **(LOC)** ، ومدة إنجاز المشروع **"T_d"** بالأشهر أو بالسنوات.

وقد اقترح **Putnam** مجموعة من المعادلات المشتقة من النموذج الأصلي لتبسيطه ولتصبح أكثر عمومية ، فتم تعريف زمن التطوير الأصغر **(T_{min})** بالمعادلة رقم (9) ، والجهد المقابل له بالمعادلة رقم (10).

$$T_{min} = 8.14 * (S / P)^{3/7} \quad \text{Month} \ldots\ldots\ldots\ldots\ldots\ldots (9)$$

$$E = 56.4 * B * (S / P)^{9/7} \quad \text{Person-Month} \ldots\ldots\ldots\ldots (10)$$

تدريب (2)



احسب الجهد اللازم لتطوير برمجية عدد سطورها (LOC = 200000) ، الثابت "C = 4000" ، فترة تطوير المشروع سنتان (T_d = 2).

جدول رقم 7.2 : قيم عامل الخبرات الخاصة **(B) كدالة في حجم البرمجية (SLOC)**

([19] عن [18])

عامل الخبرات الخاصة **(B)**	**حجم البرمجية (SLOC)**
0.16	5-15K
0.18	20K
0.28	30K
0.34	40K
0.37	50K
0.39	>70K

جدول رقم 7.3 : قيم عامل الإنتاجية**(P) كدالة في نوع التطبيق البرمجي ([19] عن [18])**

Software Application Type **نوع التطبيق البرمجي**	**Productivity Parameter Value** **قيمة عامل الإنتاجية**
Microcode	987
Firmware (ROM)	1,597
Real-Time Embedded, Avionics	1,974
Radar Systems	3,194
Command and Control	4,181
Process Control	5,186
Telecommunications	8,362
Systems Software, Scientific Systems	13,530
Business Systems	28,657

- مميزات نموذج **Putnam** :

يمتاز نموذج Putnam بالتالي :

- يحتوي على مجموعة مكثفة من أدوات إدارة تطوير المشروع والتي تدعم جميع مراحل تطوير المنتج البرمجي.
- يستخدم البرمجة الخطية **(Linear Programming)** والتي تسمح للمستخدم

بوضع قيود على عملية التطوير مثل : فترة تنفيذ المشروع القصوى (**Maximum Schedule**) ، أقل وأقصى عدد لفريق تطوير المشروع (**Low and High Bounds of Staffing**) ، الحد الأدنى والأقصى للتكلفة (**Minimum and Maximum Cost**). كما تستخدم المحاكاة الإحصائية (**Statistical Simulation**) مثل : **Monte Carlo** مما يزيد من دقة التقدير.

- يسمح للمثمن بإجراء معايرة دقيقة (**Fine Tune Calibration**) باستخدام قاعدة بيانات تاريخية لمشاريع سابقة مطورة في نفس بيئة التطوير ، كما يسمح للمثمن بتقدير حجم البرمجية (**Software Sizing**) باستخدام منهجيات متضمنة بداخله ، كما يسمح له بإدخال جميع الخصائص التي تؤثر على تكلفة المشروع.
- تدعم "**What-if Analysis**" (ماذا إذا؟) ، بمعنى ماذا يحدث إذا تم تغيير خاصية من خصائص البرمجية ، وما تأثير ذلك على التكلفة وتطوير المشروع؟ ، وبالتالي يكون من السهل تغيير البيانات المدخلة (**Input Data**).
- يُخرج النموذج وقت التطوير الأدنى (**Minimal Time**) مصحوباً بالجهد المطلوب والتكلفة ، بالإضافة إلى تحليل لحساسيتها في حالة تغيير العوامل المؤثرة فيها لقيم انحراف معياري (**Standard Deviation**) تتراوح من (1) إلى (3).

- عيوب نموذج **Putnam** :

ومن أهم عيوب نموذج Putnam :

- يعمل بطريق جيدة في حالة المشاريع ذات الحجم الكبير (الحجم أكبر من 5000 سطر ، والجهد أكبر من 1.5 شخص ــ شهور ، وفترة التطوير لا تقل عن 6 شهور).
- يجب معرفة حجم البرمجية قبل استخدام النموذج.
- قيمة التقدير حساسة بدرجة كبير إلى :
  - قيم الثوابت الداخلة في تركيبه (**C or B and P**).
  - قيمة فترة تنفيذ المشروع (T_d).
  - قيمة حجم البرمجية.

- يوجد العديد من العوامل التي يجب أن تؤخذ في الاعتبار لتضمين ظروف بيئة التطوير بطريقة جيدة في عملية التقدير ، بعض من هذه العوامل من الصعب تحديدها.
- يستخدم نموذج الشلال **(Waterfall Model)** لمحاكاة دورة حياة المنتج البرمجي والذي لا يأخذ في الاعتبار باقي نماذج التطوير مثل النموذج الحلزوني **(Spiral Model)** .
- النموذج إلى حد ما غير مفتوح ويميل إلى الملكية الشخصية **(Sometimes Algorithms may be Proprietary)** .
-

أسئلة تقويم ذاتي

ما هي أهم المعايير التي تستخدم لتقويم نماذج تقدير تكلفة المشاريع البرمجية؟.

لبوتنام نمذجان أحدهما يعتمد على نموذج دورة حياة البرمجيات والآخر يعتمد على الإنتاجية.

أ) ماهو الشكل الرياضي الذي يمثل كل نموذج.

ب) عرف الثوابت لكل نموذج وكيف يمكن حسابها.

ج) ما هي أهم مميزات وعيوب نموذج بوتنام؟.

2.3 نموذج بوهم (نموذج التكلفة الاستنتاجي)

[1 , 3, 20 , 23, 24, 25, 26]:

Boehm Model (COnstructive COst MOdel " COCOMO ")

نموذج كوكومو نموذج تجريبي مستنتج من التحليل الانحداري لبيانات مشاريع عديدة وخبرات تطوير سابقة ، وقد قام بتطويره **Boehm** عام 1981 م ، وصدر الإصدار الأول منه في نفس العام وأطلق علية نموذج **COCOMO I** ، أو **COCOMO 81** ، وتم بعد ذلك تحسينه حتى صدر منه الإصدار الثاني المحسن

COCOMO II في عام 2000م ، حيث يأخذ في الاعتبار المنهجيات المختلفة في تطوير البرمجيات وإعادة الاستخدام وغيرها من العوامل التي تؤثر على تكلفة المنتج البرمجي. لذا يُعد نموذج **Boehm** بنية هرمية لنماذج تقدير تكلفة المنتج البرمجي تحمل اسم نموذج كوكومو.

ونموذج كوكومو مفتوح وموثق جيداً ومتاح للجميع ، ولكنه معقد بعض الشئ ، ويُعد أشهر نموذج لتقدير تكلفة المشاريع البرمجية في الوقت الحالي.

- نموذج كوكومو الأصلي **(COCOMO I Model)** :

يتضمن نموذج كوكومو الأصلي **(COCOMO I)** مجموعة من النماذج الفرعية تستخدم المعادلتين التاليتين (معادلة رقم (11) ، والمعادلة رقم (12)) :

$$E = a * Size^{b} \quad \text{.....(11)}$$

$$TDEV = 2.5 * E^{c} \text{... .....(12)}$$

حيث :

"E" : يمثل جهد التطوير مقيساً بشخص ــ شهور.

"Size" : يمثل حجم البرمجية مقيساً بعدد سطور الشيفرة بالكيلو **(KLOC)**.

" TDEV " : يمثل الوقت اللازم لتطوير البرمجية بالشهور.

"a" ، "b" ، "c" : عوامل **(Coefficients)** تعتمد على شكل التطوير **(Development Mode)** ، وكذلك على مستوى النموذج **(Model Level)**.

ويوجد ثلاثة أشكال من التطوير **(Three Development Modes)** هما :

❶**عضوي أو متناسق الأجزاء أو مترابط (Organic)** : يعني أن المشروع البرمجي تحت التطوير صغير الحجم نسبياً ، وبسيط ، ومفهوم جيداً ، ويطور بواسطة فريق صغير ذي خبرة جيدة في مجال تطبيق المشروع ، ويحتاج قليلاً من الابتكار أو الفكر الجديد **(Little Innovation)** ، ولا توجد

قيود صارمة على وقت التسليم **(No Tight Deadline)** ، وكذلك يطور في بيئة تطوير مستقرة **(Stable Development Environment)**.

❷**شبه مترابط (Semi-Detached)** : يعني أن المشروع البرمجي تحت التطوير : متوسط من ناحية الحجم والتعقيد ، ويطور بواسطة فريق ذي خبرة متوسطة في مجال تطبيق المشروع ، ويحتاج شياً متوسطاً من الابتكار أو الفكر الجديد **(Medium Innovation)** ، وتوجد قيود متوسطة على وقت التسليم **(Medium Constrain for Deadline)** ، وكذلك يطور في بيئة تطوير متوسطة التعقيد والإمكانيات **(Medium Environment Development)**.

❸**مضَّمن (Embedded)** : يعني أن المشروع البرمجي تحت التطوير : كبير الحجم ومعقد ، حيث تكون البرمجيات جزءاً من ارتباط شديد التعقيد مع العتاد وإجراءات التشغيل ، ويطور بواسطة فريق ذي خبرة محدودة في مجال تطبيق المشروع ، ويحتاج إلى ابتكار أو فكر جديد **(High Innovation)** ، وتوجد قيود صارمة على وقت التسليم **(Tight Deadline)** ، وكذلك يطور في بيئة تطوير معقدة من حيث العتاد وواجهات المستخدم.

وكذلك يوجد ثلاثة مستويات للنموذج **(Three Model Levels)** هي :

①**أساسي (Basic)** : حيث يتم حساب الجهد وتكلفة المشروع البرمجي كدالة في حجم المشروع معبراً عنه بعدد أسطر الشيفرة المقدرة ، والجدول التالي **[20]** يوضح قيم العوامل **"a"** ، **"b"** ، **"c"** لهذا المستوى لمختلف أشكال التطوير.

Basic COCOMO	a	b	c
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

②متوسط (Intermediate) : حيث يتم حساب الجهد وتكلفة المشروع البرمجي كدالة في حجم المشروع معبراً عنه بعدد أسطر الشيفرة المقدرة ، ومجموعة من موجهات التكلفة **(Cost Drivers)** الممكن تجميعها وتوزيعها على أربع فئات أساسية سمات المنتج ، سمات العتاد ، سمات فريق التطوير ، وأخيراً سمات المشروع ، حيث تقدر هذه السمات بـ (15) سمة تُقدّر كل منها من سلم ذي ست علامات تقع بين "منخفض جداً" و"عال جداً جداً" في الأهمية والقيمة ، كما هو موضح بالجدول رقم 7.4 . وبناءً على تقديرات هذه السمات يجري تحديد عامل ضبط للجهد **(Effort Adjustment Factor "EAF")** بضرب عوامل الضرب السابقة ، ويأخذ نموذج كوكومو المتوسط الصيغة التالية للجهد (المعادلة رقم 13) :

$$E = a * Size^{b} * EAF \ldots\ldots\ldots\ldots (13)$$

والجدول التالي **[20]** يوضح قيم العوامل **"a"** ، **"b"** ، **"c"** لهذا المستوى لمختلف أشكال التطوير.

Intermediate COCOMO	a	b	c
Organic	**3.2**	1.05	0.38
Semi-detached	**3.0**	1.12	0.35
Embedded	**2.8**	1.20	0.32

تلاحظ اختلاف قيم العامل **"a"** للمستوى الأساسي عنه للمستوى المتوسط ، مع احتفاظ باقي العوامل بنفس القيم.

جدول رقم 7.4 : قيم موجهات التكلفة في نموذج كوكومو81 [21 ، 22 ، 23]

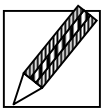
cost drivers used in the standard COCOMO model and the Effort Multipliers associated with different ratings

Category الفئة	Cost Driver موجه التكلفة	Very Low منخفض جداً	Low منخفض	Nominal عادي	High عالٍ	Very High عالٍ جداً	Extra High عالٍ جداً جداً
Product Attributes سمات المنتج	RELY اعتمادية النظام المطلوب	0.75	0.88	1.00	1.15	1.40	-
	DATA حجم قاعدة البيانات المستخدمة	-	0.94	1.00	1.08	1.16	-
	CPLX مدى تعقيد المنتج البرمجي	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes سمات الكمبيوتر	TIME القيود المفروضة على وقت التنفيذ	-	-	1.00	1.11	1.30	1.66
	STOR القيود المفروضة على سعة الذاكرة	-	-	1.00	1.06	1.21	1.56
	VIRT مدى تغيير بيئات التشغيل الخاصة بالعتاد المستخدم	-	0.87	1.00	1.15	1.30	-
	TURN المدى الزمني الذي	-	0.87	1.00	1.07	1.15	-

Category الفئة	Cost Driver موجه التكلفة	Very Low منخفض جداً	Low منخفض	Nominal عادي	High عالٍ	Very High عالٍ جداً	Extra High عالٍ جداً جداً
	يدور حوله استخدام العتاد						
Personnel Attributes السمات الشخصية	ACAP قدرة محلل المشروع	1.46	1019	1.00	0.86	0.71	-
	AEXP مدى الخبرة بالتطبيقات	1.29	1.13	1.00	0.91	0.82	-
	PCAP قدرة المبرمج	1.42	1.17	1.00	0.86	0.70	-
	VEXP الخبرة بالعتاد وخصوصاً نظم التشغيل	1.21	1.10	1.00	0.90	-	-
	LEXP الخبرة بلغات البرمجة	1.14	1.07	1.00	0.95	-	-
Project Attributes سمات المشروع	MODP مدى استخدام خبرات البرمجة الحديثة	1.24	1.10	1.00	0.91	0.82	-
	TOOL استخدام أدوات البرمجيات	1.24	1.10	1.00	0.91	0.83	-
	SCED ضغط جدول مواعيد التطوير	1.23	1.08	1.00	1.04	1.10	-

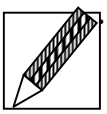
③**متقدم (Advanced)** : حيث يتم حساب الجهد وتكلفة المشروع البرمجي كدالة في حجم المشروع معبراً عنه بعدد أسطر الشيفرة المقدرة ، ومجموعة من موجهات التكلفة **(Cost Drivers)** مثل النموذج المتوسط ، إضافة إلى السماح بضبط موجهات التكلفة لكل مرحلة من مراحل التطوير، وكذلك إجراء بعض التعديلات على مستوى وحدة البرمجة المركبة **(Module)** ، وعلى مستوى النظام **(System Level)**.

تدريب (2)



مشروع شبه مترابط (Semi-Detached) حجمه 32KLOC أحسب الجهد المطلوب (Effort) والفترة الزمنية (Schedule) ، وكذلك الإنتاجية (Productivity) ، وكذلك متوسط عدد فريق العمل (Average Stuffing) المطلوبين لتطوير هذا المشروع ، باستخدام نموذج كوكومو الأساسي.

تدريب (3)



استخدم نموذج كوكومو المتوسط في حل التدريب السابق ، علماً بأن المشروع تحت التطوير ذات تعقيد عالٍ (High Complexity) ، ويحتاج لتنفيذه ذاكرة تخزين عالية (High Memory Capacity) وفريق التطوير ذات خبرة ضئيلة في التطبيقات وكذلك قدرة المبرمجين ضئيلة ، ويحتاج إلى قاعدة بيانات ذات حجم عالٍ جداً ، أما باقي موجهات التكلفة (السمات) فهي عادية.

◆ نموذج كوكومو الثاني **(COCOMO II Model)** :

يفترض نموذج **COCOMO 81** أن البرمجية سوف تطور طبقاً لنموذج الشلال مستخدماً اللغات الأمرية العادية مثل : لغة **C** ، ولغة **FORTRAN**. ولكن في الوقت الراهن يوجد تغيير جوهري في تقنيات ومنهجيات ولغات تطوير البرمجيات مقارنة بما كان عليه الحال في وقت ظهور هذا النموذج ، حيث إنه حالياً تطور البرمجيات غالباً باستخدام لغات برمجة قواعد البيانات مثل **SQL** ، وكذلك نظم قواعد البيانات التجارية **(Commercial Database Management System)** ، مستخدماً في كثير من الأحيان المكونات المعادة للاستخدام **(Reusable Components)** . هذا بالإضافة إلى استخدام الهندسة العكسية **(Re-Engineering)** لتطوير برمجيات جديدة من برمجيات موجودة بالفعل مع وجود الدعم الكامل لجميع مراحل تطوير البرمجيات من خلال استخدام الـ **CASE Tools** . لذلك تم تطوير هذا النموذج إلى نموذج **COCOMO II** ليشمل هذه التغييرات في عملية تقدير تكلفة المشاريع البرمجية ، حيث إنه يدعم منهجيات مختلفة لتطوير البرمجيات مثل : نموذج **Prototyping** ، والنموذج الحلزوني **(Spiral Model)** ، وكذلك يدعم استخدام الهندسة العكسية في تطوير البرمجيات وإعادة استخدام المكونات البرمجية ، وبصفة عامة ويتميز هذا الموديل عن نظيره السابق بالأتي:

- الموديل السابق يتطلب معرفة مبدئية لحجم البرمجية كأساس في حين أن هذا الموديل يأخذ جهود تخمين مختلفة معتمدا على مرحلة التطوير للمشروع.
- الموديل السابق يعطي تقدير للجهد والوقت في حين أن هذا الموديل يعطي مدى للتقدير الذي يقع الجهد المطلوب في حدوده.
- الموديل الحالي يتكيف مع متطلبات إعادة الاستخدام والهندسة العكسية **(Reengineering)** حيث تستخدم العديد من الوسائل لترجمة البرمجية الموجودة في حين أن الموديل السابق لا يأخذ في الاعتبار هذه العوامل.

- الموديل الحالي يأخذ في الاعتبار التغير السريع في المتطلبات.
- مركبة الحجم في الموديل الحالي تستخدم 5 عوامل لتعميم واستبدال تـأثيرات نموذج التطوير بدلاً من استخدام ثلاثة أشكال من التطوير **(Three Development Modes)** في النموذج السابق، كما سيتم توضيحه فيما بعد.
- تم إضافة (7) موجهات تكلفة جديدة **(7 New Cost Drivers)** هي : **DOCU, RUSE, PVOL, PLEX, LTEX, PCON, SITE** ، وتم حذف (5) موجهات تكلفة في النموذج السابق هي : **VIRT, TURN, VEXP, LEXP, MODP** ، كما سـيتم توضيحه فيما بعد.

يتضمن نموذج **COCOMO II** سلسلة من أربعة نماذج فرعية متتالية تسمح بتقدير تكلفة المشاريع البرمجية بناءً على التفاصيل المتاحة لحسـاب التقـديرات ، وهي كما يلي:

① نموذج تكوين التطبيق (The Application Composition Model) :

وهو يدعم تقدير تكلفة المشـاريع المطـورة باسـتخدام نمـوذج العمليـات **Prototyping** ، وكذلك المشاريع المطورة عن طريق تجميـع وتهيئـة مكونـات برمجية موجودة ، والتي تستخدم تقنية إعادة الاسـتخدام بكثافـة. ويسـتخدم هـذا النموذج نقاط الكائن **(Object Points)** لتقدير الجهد اللازم لعملية التطوير باستخدام معادلة بسيطة (المعادلة رقم 14) ، وهي :

$$E = NOP * (1 – Reuse/100) / PROD \quad \text{.....(14)}$$

حيث : **"E"** : الجهد **(Effort)** مقيس بالشـخص ــ شـهور **(Person-Month)** ، **"NOP"** : عـدد نقـاط الكـائن **(Object Points)** ، و **"PROD"** : الإنتاجيـة **(Productivity)** ، وتقدر كما هو موضـح بالجـدول رقـم 7.5 ، و**"Reuse"** : النسبة المئوية لنقاط الكائن التي سيعاد استخدامها من مشاريع سابقة.

جدول رقم 7.5 : قيم معامل إنتاجية نقاط الكائن كدالة في إمكانات وخبرة المطور وأدوات وإمكانات ICASE [23]

Very High	High	Nominal	Low	Very Low	Developer's Experience and Capability إمكانات وخبرة المطور
Very High	High	Nominal	Low	Very Low	ICASE Maturity and Capability نضج وإمكانات ICASE
50	25	13	7	4	معامل الإنتاجية "PROD"

ويتم تحديد الحجم الابتدائي للتطبيق البرمجي بإحصاء عدد الشاشات **(Screens)** ، والتقارير **(Reports)** ، والمكونات البرمجية المكتوبة بلغات الجيل الثالث **(Third-Generation Components)** والتي سوف تستخدم في التطبيق ، ومن ثم يتم تحديد مدى تعقيد الكائن **(Object Complexity)** مستخدماً الإرشادات الموضحة بالجداول التالية (7.6 , 7.7 ,7.8) **([23])**.

جدول رقم 7.6 : مستويات تعقيد نقاط الكائن للشاشات [23]
Object Points Complexity Level for Screen

Number and Source of Data Tables			Number of Views Contained
8+>Total	8>Total	4>Total	
Medium	Simple	Simple	3>
Difficult	Medium	Simple	7−3
Difficult	Difficult	Medium	8+

جدول رقم 7.7 : مستويات تعقيد نقاط الكائن للتقارير [23]
Object Points Complexity Level for Reports

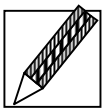
Number and Source of Data Tables			Number of Views Contained
8+>Total	8>Total	4>Total	
Medium	**Simple**	**Simple**	3>
Difficult	**Medium**	**Simple**	7−3
Difficult	**Difficult**	**Medium**	8+

جدول رقم 7.8 : وزن مستويات تعقيد نقاط الكائن [23]
Complexity Levels Weights for Object Points

Difficult	Medium	Simple	نوع الكائن
3	2	1	**Screen**
8	5	2	**Report**
10	−	−	**3GL Component**

ومن ثم حساب الحجم النهائي للتطبيق البرمجي **(NOP)** بعد أخذ مدى تعقيد الكائن في الاعتبار ، ومن ثم حساب الجهد المطلوب من المعادلة رقم (13) بعد تحديد نسبة إعادة الاستخدام **(Reuse)** ، وإنتاجية المطور والتي تعتمد على خبرة وإمكانيات المطور **(Developer's Experience and Capability)** ، وكذلك تعتمد على إمكانيات الـ **CASE Tools** المستخدمة لدعم عملية التطوير كما هو موضح بالجدول 7.5 السابق.

تدريب (4)



تم تطوير تطبيق برمجي يحتوي على : عدد (10) شاشات ذات تعقيد متوسط ، وعدد (8) تقارير ذات تعقيد متوسط ، وعدد (6) مكونات برمجية باستخدام لغات الجيل الثالث (Six 3GL Modules) ، وإعادة الاستخدام في هذا التطبيق كانت 25% ، وبيئة ICASE شبه ناضجة ومؤهلة لعملية التطوير (ICASE environment is relatively mature and capable) ، وفريق التطوير غير كفء وتُعد كفاءته منخفضة ، أحسب الجهد المطلوب لتطوير التطبيق البرمجي.

②نموذج التصميم المبكر (The Early Design Model

يستخدم هذا النموذج ، بعد الموافقة على متطلبات المشروع البرمجي تحت التطوير ومع بداية العمل في المراحل الأولية من تصميمه ، لعمل تقدير مبدئي للجهد والوقت اللازمين لتطويره ، وفيه يستخدم مجموعة جديدة من موجهات التكلفة **(Cost Drivers)** ومعادلات تقدير تعتمد على نقاط الدالة **(FP)** أو عدد أسطر الشيفرة مقدراً بالكيلو **(KLOC)**.

وتعتمد عملية التقدير في هذه المرحلة على المعادلة القياسية للنماذج الخوارزمية (المعادلة رقم (15)) ، وهي كما يلي:

$$E = a * Size^{b} * EAF \ldots\ldots\ldots\ldots (15)$$

حيث :

- **"a"** : ثابت ويساوي (2.94) بناءً على ما اقترحه مطور هذا النموذج **(Boehm)** من خلال الدراسات التي قام بها على العديد من البيانات التاريخية

لمشاريع سابقة ، ويستحسن أن يتم معايرته طبقاً لبيانات تاريخية خاصة بمشاريع سبق وأن تم تطويرها في نفس بيئة التطوير.

- **"Size"** : هو حجم البرمجية ويعبر عنه بـــ **"KSLOC" (Thousands of Lines of Source Code)** ، ويتم حسابه من خلال تقدير عدد نقاط الدالة **(FP)** في البرمجية ومن ثم تحويلها إلى **"KSLOC"** باستخدام الجداول المعيارية التي تربط حجم البرنامج **(KSLOC)** بنقاط الدالة **(FP)** لعدة لغات مختلفة ، والتي تم تناولها في الفصل الأول من هذه الوحدة.
- **"b"** : هو معامل الأس **(Exponent Term)** يعكس مدى الزيادة المطلوبة في الجهد لمقابلة الزيادة في حجم البرمجية ، وهو ليس ثابتاً كما هو الحال في نموذج **COCOMO81** بل يختلف من نظام برمجي إلى آخر ، وتحسب طبقاً للمعادلة التالية :

$$b = 0.91 + 0.01 \sum_{i=1}^{i=5} SF_i \quad \text{.....(16)}$$

ويقدر وزن هذه المعاملات الخمسة (SF_i) على مقياس من : عالية جدا جداً إلى منخفضة جداً ، مروراً بعالية جداً ، وعالية ، وعادية ، ومنخفضة وهذه المعاملات كالتالي :

1. سابقة الأعمال **("PREC" Precedentedness)** : وهو مقياس يعكس الخبرة السابقة للمنشأة المنفذة للمشروع في التعامل مع مشاريع مماثلة للمشروع القائم ، فإذا كانت سابقة الأعمال قليلة جداً فهذا يعني عدم وجود خبرة سابقة ، وفي هذه الحالة يقدر وزن هذا المقياس بالعدد **(6.2)** ، وإذا كانت سابقة الأعمال عالية جداً جداً فهذا يعني أن للمؤسسة باعاً طويلاً في تطوير مثل هذه المشاريع ، وفي هذه الحالة يقدر وزن هذا المقياس بالعدد **(0)**. كما هو موضح بالجدول رقم 7.9 ، وفي حالة وجود سابقة أعمال عالية جداً جداً فهذا يعني أن المنشأة على قدر كبير جداً من فهم وظائف ومتطلبات المنتج البرمجي ، بالإضافة إلى عدم الحاجة أو حاجتها بقدر ضئيل لابتكار هياكل وخوارزميات جديدة لمعالجة البيانات.

2. مرونة التطوير **(Development Flexibility "FLEX")** : وهو مقياس يعكس درجة المرونة في عملية التطوير ، حيث توصف عملية التطوير بعدم المرونة في حالة الحاجة إلى الالتزام الكامل من قبل الشركة المنفذة للمشروع بتنفيذ جميع مواصفات ومتطلبات المنتج البرمجي بدون أي تغيير **(FLEX = 5.07)** ، وتوصف بالمرونة الكاملة في حالة قيام العميل بوضع الأهداف العامة للمنتج البرمجي فقط وترك التفاصيل بالكامل للشركة المنفذة **(FLEX = 0)** ، كما هو موضح بالجدول رقم **7.9**.

3. دقة التصميم المعماري/تحليل المخاطر **(Architecture/Risk Resolution " RESL ")** : وهو مقياس يعكس درجة تحليل المخاطر التي تمت أثناء التصميم المعماري للمنتج البرمجي ، ويتم وضع **RESL = 0** في حالة التحليل الكامل للمخاطر ، و **RESL = 5.65** في حالة التحليل المنخفض للمخاطر ، وهكذا.... ، أما في حالة عدم تحليل المخاطر على الإطلاق أو المنخفض جداً فيتم وضع **RESL = 7.07** ، كما هو موضح بالجدول رقم **7.9**.

4. تماسك وتناغم الفريق **(Team Cohesion "TEAM")** : وهو مقياس يعكس درجة تماسك الفريق من حيث معرفتهم لبعضهم ، ومدى خبرتهم في العمل كفريق متكامل ، ومدى توافق أهدافهم وثقافتهم ولغاتهم ، ومدى القدرة والرغبة في تعاونهم وتفاعلهم مع بعضهم الى غير ذلك. ويُعد تماسك وتناغم الفريق عالياً جداً **(TEAM = 0)** في حالة وجود تكامل وفعالية بين فريق العمل بدون مشاكل اتصالات ، ومنخفض جداً **(TEAM = 5.48)** في حالة عدم وجود تكامل وتفاعل وصعوبة في الاتصالات بين أعضاء الفريق، كما هو موضح بالجدول رقم 7.9.

5. كمال العملية **(Process Maturity "PMAT")** : وهو مقياس يعكس مدى نضج واكتمال عمليات المؤسسة ، ويمكن التعامل معه طبقاً للقيم الواردة بالجدول رقم 7.9.

جدول رقم 7.9 : قيم وزن المعاملات الخمسة (SF) [25]

Scale Factors عوامل الوزن	قيم وزن المعاملات (Factors Weight Values "SF")					
	Very Low	Low	Nominal	High	Very High	Extra High
PREC	6.2	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	3.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

- **"EAF"** : عامل ضبط الجهد **(Effort Adjustment Factor)** لضبط الجهد ليتضمن تأثير موجهات التكلفة **(Cost Drivers)** ، ويشمل هذا النموذج عدد (7) موجهات للتكلفة موضحة بالجدول 7.10 مع نظائرها من النموذج المعماري اللاحق **(Post-Architecture Model)** ، والموضحة في الجدول 7.11 . وتقدر قيمة **"EAF"** من المعاداة التالية :

$$EAF = \prod_{i=1}^{i=n} EM_i \quad \text{.....(17)}$$

حيث **"EM_i"** هو قيمة مضروب الجهد لموجهات التكلفة ، و "n" هو عدد موجهات التكلفة وتساوي (7) لنموذج التصميم المبكر و (17) للنموذج المعماري اللاحق.

جدول رقم 7.10 : العلاقة بين موجهات التكلفة لنموذج كوكومو [25]

	Early Design cost drivers	Post-Architecture cost drivers (Counterpart combined)
Product reliability and complexity	RCPX	RELY, DATA, CPLX, DOCU
Required reuse	RUSE	RUSE
Platform difficulty	PDIF	TIME, STOR, PVOL
Personnel capability	PERS	ACAP, PCAP, PCON
Personnel experience	PREX	AEXP, PEXP, LTEX
Facilities	FCIL	TOOL, SITE
Required Development Schedule	SCED	SCED

جدول رقم 7.11 : موجهات التكلفة لنموذج كوكومو (النموذج المعماري اللاحق) [25]

Cost Driver	Description	Rating					
		Very Low	Low	Nominal	High	Very High	Extra High
سمات المنتج Product							
RELY	Required Software Reliability	0.82	0.92	1.00	1.10	1.26	-
DATA	Database Size	-	0.90	1.00	1.14	1.28	-
CPLX	Product Complexity	0.73	0.87	1.00	1.17	1.34	1.74
RUSE	Required Reusability	-	0.95	1.00	1.07	1.15	1.24
DOCU	Documentation	0.81	0.91	1.00	1.11	1.23	-
سمات منصة Platform التشغيل							
TIME	Execution Time Constraint	-	-	1.00	1.11	1.29	1.63
STOR	Main Storage Constraint	-	-	1.00	1.05	1.17	1.46
PVOL	Platform Volatility	-	0.87	1.00	1.15	1.30	-

Cost Driver	Description	Rating					
		Very Low	Low	Nominal	High	Very High	Extra High
Personnel السمات الشخصية							
ACAP	Analyst Capability	1.50	1.22	1.00	0.83	0.67	-
PCAP	Programmer Capability	1.34	1.15	1.00	0.88	0.76	-
PCON	Personnel Continuity	1.29	1.12	1.00	0.90	0.81	-
AEXP	Applications Experience	1.22	1.10	1.00	0.88	0.81	-
PLEX (PEXP)	Platform Experience	1.19	1.09	1.00	0.91	0.85	-
LTEX	Language and Tool Experience	1.20	1.09	1.00	0.91	0.84	
Project سمات المشروع							
TOOL	Software Tools	1.17	1.09	1.00	0.90	0.78	-
SITE	Multisite Development	1.22	1.09	1.00	0.93	0.86	0.80
SCED	Development Schedule	1.43	1.14	1.00	1.00	1.00	-

ولكن السؤال الآن كيف يتم حساب مضروب الجهد (Effort Multiplier) لكل موجة من موجهات التكلفة السبعة من خلال التناظر مع موجهات التكلفة الخاصة بالنموذج المعماري اللاحق : للإجابة على هذا السؤال اتبع الخطوات التالية :

1. قم بإسناد قيمة رقمية على المقياس من 1 إلى 6 إلى موجهات التكلفة الخاصة بالنموذج المعماري اللاحق المقابلة لموجه التكلفة المراد حساب مضروب الجهد الخاص به طبقاً للتقديرات الخاصة بها : بمعنى إذا التقديرات الخاصة بها "**منخفضة جداً**" تعطى القيمة الرقمية **(1)** ، وإذا كانت "**منخفضة**" تعطي القيمة الرقمية **(2)** ، .. وهكذا كما هو موضح بالجدول التالي.

عال جداً جداً Extra High	عال جداً Very High	عال High	عادي Nominal	منخفض Low	منخفض جداً Very Low
6	5	4	3	2	1

2. قم بجمع هذه التقديرات والتي من خلالها يتم تقدير موجه التكلفة المراد حساب مضروب الجهد له وبالتالي معرفة مضروب الجهد من الجدول الخاص بـــه ، ونوضح ذلك بالمثال التالي :

- قــم بحســاب مضــروب الجهــد الخــاص بموجــه التكلفــة **RCPX** إذا كــان : **DOCU = Nominal ، CPLX = Nominal ، DATA = Low ، RELY = Low PVOL = Very High ، STOR = LOW ، Time = High ،**

الحل : من الجدول رقم **7.10** نرى أن موجه التكلفة **RCPX** يقابل موجهات التكلفة **RELY ، DATA ، CPLX ، DOCU** ، بإتباع الخطوات السابقة نجد أن :

1. **RELY = 2 ، DATA = 2 ، CPLX = 3 ، DOCU = 3** .
2. مجموع التقديرات = $2 + 2 + 3 + 3 = 10$
3. من الجدول الخاص بموجه التكلفة **RCPX** التالي نجد أن القيمة 10 تدل على أن **RCPX = Low** ، ومضروب الجهد الخاص به هو : 0.83.

الجداول الخاصة بقيم مضروب الجهد لموجهات التكلفة السبع **[25]**

RCPX Cost Driver

RCPX Descriptors:							
• Sum of RELY, DATA, CPLX, DOCU Ratings	5, 6	7, 8	9 - 11	12	13 - 15	16 - 18	19 - 21
Rating Levels	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.49	0.60	0.83	1.00	1.33	1.91	2.72

PDIF Cost Driver

PDIF Descriptors:					
• Sum of TIME, STOR, and PVOL ratings	8	9	10 - 12	13 - 15	16, 17
Rating Levels	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.87	1.00	1.29	1.81	2.61

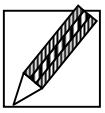
PERS Cost Driver

PERS Descriptors:							
• Sum of ACAP, PCAP, PCON Ratings	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Rating Levels	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	2.12	1.62	1.26	1.00	0.83	0.63	0.50

PERS Cost Driver
FCIL Cost Driver

FCIL Descriptors:							
• Sum of TOOL and SITE ratings	2	3	4, 5	6	7, 8	9, 10	11
Rating Levels	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.43	1.30	1.10	1.0	0.87	0.73	0.62

تدريب (5)



تم تطوير تطبيق برمجي يحتوي على : عدد (10) شاشات ذات تعقيد متوسط ، وعدد (5) شاشات ذات تعقيد عالٍ ، وعدد (8) تقارير ذات تعقيد متوسط ، وعدد (12) مكونات برمجية باستخدام لغات الجيل الثالث (Six 3GL Modules) ، وقد تم استخدام إحدى لغات البرمجة كائنة المنحى (Object Oriented Programming Language) ، أحسب الجهد المطلوب لتطوير

③نموذج إعادة الاستخدام (Reuse Model) [25]

تُعرف الشيفرة الجديدة **(New Code)** في نموذج **COCOMO II** : بأنها الشيفرة المكتوبة بطريقة يدوية بدون استخدام أي من الأدوات البرمجية **(CASE Tools)** ، ما عدا ذلك يُعد استخدام أي شيفرة نوعاً من أنواع إعادة الاستخدام ، وفي هذه الحالة يستخدم نموذج إعادة الاستخدام لتقدير الجهد اللازم لدمج **(Integrate)** الشيفرة المعادة للاستخدام **(Reused Code)** أو المولدة من الأدوات البرمجية المساعدة **(Generated Code)** مع الشيفرة الجديدة ، ويمكن حصر أنواع أو صور إعادة الاستخدام فيما يلي :

- الشيفرة معادة الاستخدام **(Reused Code)** : وهي شيفرة موجودة مسبقاً **(Pre-Existed Code)** وتم استخدامها كما هي بدون أي تغيير وبدون محاولة لفهمها ، ويطلق عليها أيضاً شيفرة الصندوق الأسود **(Black Box Code)** ، والجهد

المطلوب في هذه الحالة يساوي الصفر.

- الشيفرة المعاد تهيئتها **(Adapted Code)** : وهي شيفرة موجودة مسبقاً وتم تعديلها لتدمج مع باقي المكونات البرمجية للمنتج البرمجي ، ويطلق عليها أيضاً شيفرة الصندوق الأبيض **(White Box Code)** ، وفي هذه الحالة يوجد جهد لفهم الشيفرة قبل تعديلها وجهد لتعديلها حتى تعمل بنجاح مع باقي المكونات البرمجية.
- المكونات التجارية الجاهزة **(Commercial off-the-shelf "COTS" Components)** : وهي شيفرة تستخدم عن طريق التأجير أو عن طريق الترخيص باستخدامها **(Leased or Licensed Source Code)**.
- الشيفرة المعاد تحويلها آلياً **(Automatically Translated Code)** : وهي شيفرة موجودة مسبقاً ويتم تحويلها بطريقة آلية باستخدام طرق الهندسة العكسية **(Reengineering)** باستخدام الأدوات البرمجية المؤتمتة **(Automated Software Tools)**. وتُعد تضمين الجهد المطلوب لتنفيذ ذلك في الجهد الكلي المطلوب لتطوير البرمجية من التحسينات الإضافية في موديل **COCOMO II** ، وهذا الجهد الإضافي يتم حسابه من المعادلة رقم (18) :

$$E_{Auto} = ASLOC * (AT/100) / ATPROD \ldots\ldots\ldots(18)$$

حيث **"ATPROD"** : إنتاجية المطورين في دمج الشيفرة المعاد تحوليها آلياً مع باقي مكونات المنتج البرمجي وتساوي 2400 كما اقترحها مطور النموذج **(COCOMO II Manual)** ، و **"ASLOC"** : حجم المكونات البرمجية (الشيفرة) التي تم تهيئتها **(Size of Adapted Code)** ، و **"AT"** النسبة المئوية لحجم الشيفرة المعاد تحويلها آلياً.

- أسطر الشيفرة المكافئة **(Equivalent Source Line of Code "ESLOC")** : وهي أسطر الشيفرة التي يتم إضافتها إلى حجم البرمجية الأصلي لتضمين الجهد المبذول في تعديل وتضمين الشيفرات المعاد استخدامها **(Reused**

(Code أو تلك التي يعاد تهيئتها (Adapted Code) ، ويمكن تقديرها باستخدام المعادلة رقم (19) :

$$KESLOC = KASLOC * (1 - AT/100) * AAM \ldots\ldots\ldots (19)$$

حيث : "KESLOC " : حجم الشيفرة المكافئة مقاسة بالكيلو ، وتلاحظ نقصان حجم الشيفرة المكافئة كلما زاد حجم الشيفرة المولدة آلياً. و "AAM" يمثل معامل ضبط لعملية تهيئة المكونات المعادة للاستخدام للاندماج بصورة صحيحة مع باقي مكونات المنتج البرمجي ، وتحسب من خلال المعادلة رقم (20) :

$$AAM = (AA + SU + 0.4DM + 0.3CM + 0.3IM) / 100 \ldots (20)$$

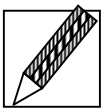
حيث إن : "AA" : معامل يعكس الجهد المطلوب لاختيار وتحليل المكونات المفترض إعادة استخدامها ، حيث الجهد المطلوب لإعادة استخدام شيفرة معينة لا يبدأ من الصفر حتى إن لم يتم إعادة استخدامها ، حيث إنه تم بذل مجهود لتقرير مدى صلاحية استخدامها من عدمه ، وتتراوح قيمة هذا المعامل من (0) إلى (8) حسب الجهد المطلوب في عمليتي الاختيار والتحليل.

"SU" : معامل يعكس الجهد المطلوب لفهم المكونات البرمجية المعادة للاستخدام وتعتمد قيمته على مدى تعقيد الشيفرة من حيث الهيكل (Structure) ، ودرجة وضوح التطبيق (Application Clarity) ، ومدى الشرح والوصف الذاتي للشيفرة (Self-Descriptiveness) ، وكذلك تعتمد على مدى دراية وألفة المطورين بهذه المكونات ، وتتراوح قيمة هذا المعامل من (50) للشيفرات غير المهيكلة والمعقدة إلى (10) للشيفرات المكتوبة بصورة هيكلية منظمة وعلى صورة مكونات برمجية مركبة (Strong Modularity) وموثقة توثيقاً جيداً. مع ملاحظة استخدامها فقط في حالة وجود مكونات قد تم إعادة استخدامها بالفعل.

"DM" ، "CM" ، "IM" : وهي على التوالي : النسبة المئوية للتصميم المعدل (Percent Design Modified) ، النسبة المئوية للشيفرة المعدلة (Percent Code

(Modified) ، والنسبة المئوية للمجهود المطلوب لدمج مكونات إعـادة الاسـتخدام **(Percent of Integration Required for Adapted Software)**.

تدريب (6)



في التدريب (4) تم إضافة 8KSOL عن طريق إعادة الاستخدام منها 45% مولدة بطريق آلية ، وبفرض عدم وجود أية تغييـرات جوهرية في قيم عوامل الأس أو فـي قـيم موجهـات التكلفـة ، وبفرض أنه تم تعديل ما نسبته 40% في التصـميم ، و 60% في الشيفرة ، و 80% في المجهود المطلوب للـدمج ، والعامـل الذي يعكس الجهد المطلوب في عمليتي اختيار وتحليل المكونـات المعادة للاستخدام يمكن أن يقدر بـ (5%) ، والعامل الذي يعكس الجهد المطلوب لفهم المكونات البرمجية المعادة للاستخدام يمكـن أن يقدر بـ (20%). أحسب الجهد الإضافي المطلوب.

④النموذج المعماري اللاحق

(Post - Architecture Model) [25] :

يستخدم هذا النموذج نفس معادلات التقديرات في نموذج التصـــميم المبكـــر بالإضافة إلى معادلات إعادة الاستخدام الواردة في نموذج إعادة الاســـتخدام ، هـــذا بالإضافة إلى عامل جديد **(REVL)** لضبط الحجم الفعال **(Effective Size)** للمنـــتج البرمجي نتيجة تطور وعدم ثبـــات المتطلبـــات **Requirements Evolution and (Volatility)** أثناء مراحل التطوير المختلفة ، وذلك باستخدام المعادلة رقم (21) :

Effective Size = Original Size (1 + REVL/100) …. (21)

حيث **REVL** : هو عامل لضبط الحجم نتيجة تطور وعدم ثبات المتطلبات ، وهـــو عبارة عن نسبة مئوية من الحجم الأصلي للمنتج البرمجي.

المعادلات التالية تمثل نموذج **COCOMO II** في صورته النهائية :

$$E = a * Size^{b} * EAF + E_{Auto}$$

a = 2.94 (يجب معايرته بيانات مشاريع سابقة مطورة في نفس بيئة التطوير)

$$b = 0.91 + 0.01 \sum_{i=1}^{i=5} SF_i$$

$$EAF = \prod_{i=1}^{i=n} EM_i$$

حيث "n" = 7 لنموذج التصميم المبكر ، 17 للنموذج المعماري اللاحق

$$E_{Auto} = ASLOC * (AT/100) / ATPROD$$

$$Size = (KSLOC + KASLOC * (1 – AT/100) * AAM)(1+REVL/100)$$

- تقدير فترة تطوير المشروع في نموذج **COCOMO II [25]** :

(Schedule Estimation in COCOMO II Model) :

يتم تقدير فترة تطوير المشروع البرمجي في نمـــوذج **COCOMO II** فـــي

صورته المبسطة من خلال المعادلة رقم **(22)** :

$$T_{DEV} = [C * E_{NS}^{(D + 0.2*(B - 0.91))}] * SCED\% / 100 \ldots (22)$$

حيث : C = 3.67 ، D = 0.28 ، ولكن يجب معايرتهما طبقاً لبيانات مشاريع سبق وأن تم تطويرها في نفس بيئة التطوير الحالية. أما عامل الضرب **"SCED"** فيمثل النسبة المئوية لضغط أو زيادة فترة التطوير ، و **"E_{NS}"** فهو الجهد الكلي اللازم لعملية التطوير مع وضع عامل ضرب الجهد **"SCED"** يساوي واحد **(Nominal Value)** أثناء تقديره .

أسئلة تقويم ذاتي

1. ما هي أهم المميزات التي يتميز بها نموذج كوكومو الثاني عن نظيره الأول؟.
2. "يتضمن نموذج COCOMO II سلسلة من أربعة نماذج فرعية متتالية تسمح بتقدير تكلفة المشاريع البرمجية بناءَ على التفاصيل المتاحة لحساب التقديرات" ، اكتب المعادلات التي تمثل هذه النماذج وبين كيف يمكن حساب ثوابتها؟.

4. أدوات تقدير تكلفة المشروعات البرمجية [27] :

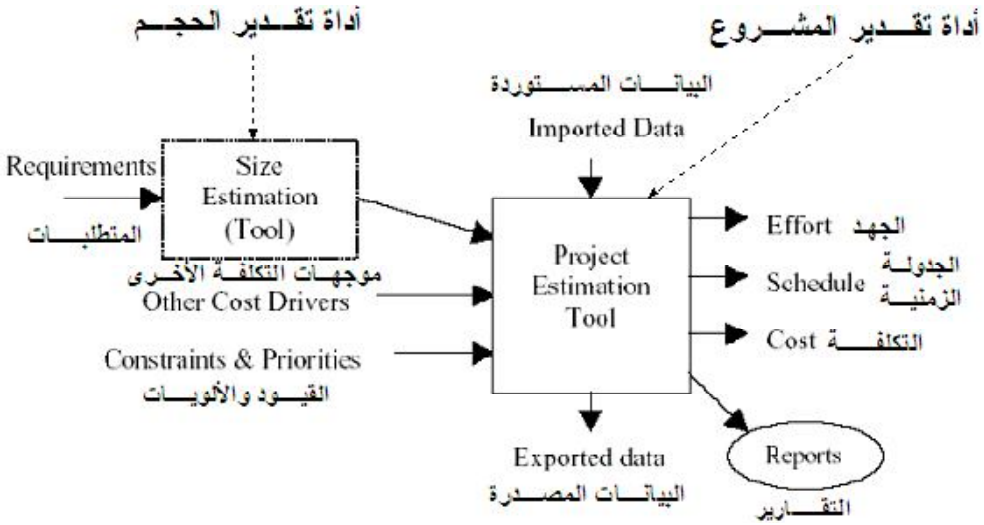
Software Projects Cost Estimation Tools :

أدوات تقدير تكلفة المشاريع البرمجية يمكن أن تكون منتجات برمجية بمفردها أو ضمن منتجات برمجية فعالة ومتكاملة لإدارة المشاريع البرمجية الكبيرة. وأدوات التقدير البرمجية يمكن أن تدعم عملية تقدير الحجم ، أو عملية تحويل الحجم إلى جهد ، وجدول زمني ، وتكلفة أو الاثنين معاً.

ليس هناك أدوات تقدير برمجية بدقة مطلقة لحل مشكلة تقدير تكلفة المشاريع البرمجية ، ولكنها في نفس الوقت تُعد وسيلة مفيدة ضمن طاقم الأدوات الخاص بعملية التقدير ، ولابد أن يؤخذ في الاعتبار أهمية استخدام أداة أو أكثر في عملية التقدير ، مع معرفة أن دقة مخرجات هذه الأدوات تعتمد على مدخلاتها والتي تعتمد بدورها على دقة تحديد حجم المشروع البرمجي وباقي العوامل المذكورة سابقاً. ويجب ــ أيضاً ــ الحذر من بائعي تلك الأدوات البرمجية والذين يدعون أن نسبة التقدير قد تصل إلى +/- نسبة ضئيلة في المائة. لذا يجب التدقيق والتأكد من صلاحية هذه الأدوات البرمجية قبل شرائها. والشكل 7.2 يوضح تخطيط مدخلات ومخرجات أدوات تقدير تكلفة المشاريع البرمجية القائمة بذاتها .

بصفة عامة يوجد العديد من السمات والشروط التي يجب أن تؤخذ بعين الاعتبار عن تقويم أدوات تقدير تكلفة المشاريع البرمجية **(Software Projects Estimation Tools Evaluation)** ، والتي من أهمها ما يلي :

1. السعر **(Price)** :



شكل 7.2 : تخطيط مدخلات ومخرجات أدوات تقدير تكلفة المشاريع البرمجية [27]

أدوات التقدير التجارية **(Commercial Estimation Tools)** يمكن أن تصنف من حيث طريقة استغلالها إلى نوعين : الأول عن طريق الإيجار السنوي **(Annual Rent)** مقابل دفع مبلغ مقطوع سنوياً **(Annual Fee)** ، الثاني: عن طريق الشراء الكامل مقابل دفع مبلغ لمرة واحدة **(One Time Fee)**. وفي الوقت الراهن يوجد ثلاث فئات من أسعار هذه الأدوات يمكن تصنيفها إلى : فئة يمكن أن يصل سعرها إلى ما يقرب من 1000 دولار أو أقل ، فئة متوسطة الثمن يمكن أن يصل سعرها إلى ما يقرب من 5000 دولار ولا يقل عن 1000 دولار ، فئة عالية الثمن يمكن أن يصل سعرها إلى 20000 دولار أو أعلى ولا يقل عن 5000 دولار.

واعتمادا على حجم المؤسسة وحجم ونوعية المشاريع التي تقوم بتنفيذها ومدى الدقة المطلوبة في عملية التقدير يتم اختيار الفئة المناسبة ، علماً بأن معظم الأدوات التي يقل سعرها عن 1000 دولار أو يساويها ــ غالباً ــ لا تحتوي على العديد من موديلات التقدير المعقدة ذات الدقة المقبولة في عملية التقدير وينقصها الكثير من الوظائف والخيارات ، ولكن يمكن الاعتماد عليها في توليد تقديرات كافية وكأداة مساعدة.

2. منصة التشغيل والأداء **(Platform and Performance) :**

يجب أن يتم معرفة العديد من العوامل التي تتعلق بأداة التقدير من حيث ما يلي :

- مدى توافقها مع منصة التشغيل والتجهيزات المادية والبرمجية الحالية الموجودة في مؤسستك أو شركتك.
- مدى توافقها مع منصات التشغيل الأخرى ، وهل يمكن أن تعمل مع العديد منها؟.
- مدى الموارد التي تحتاجها أثناء عملية التشغيل مثل : سعة الذاكرة الرئيسة **(RAM Capacity)** ، حجم القرص الصلب **(Hard Disk Space)**.
- مدى حجم قاعدة البيانات الملحقة بها ، هل تستوعب البيانات التاريخية المراد حفظها؟ ، وكذلك هل تستوعب حجم وعدد المشاريع التي سوف تتعامل معها الشركة والمؤسسة خلال الفترة المتوقع استخدامها فيها؟ .

3. سهولة الاستخدام والتوثيق **(Easy of Use and Documentation) :**

تُعد سهولة الاستخدام والتوثيق الجيد من أهم السمات التي يجب أخذها في الاعتبار عند شراء أي برمجية من البرمجيات ، فمثلاً يجب أن تكون قادراً على استخدام الأداة في عملية التقدير بسهولة ويسر بدون الدخول في التفاصيل الخاصة بالأداة ، وفي حالة طلب المساعدة تجد ذلك بسهولة وبتوضيح تام في دليل الاستخدام الخاص بالأداة ، والذي يجب أن يوضح كيفية استخدام الأداة لتوليد تكلفة المشروع ،

وكذلك يجب أن يحتوي أمثلة توضيحية ، وعلى شرح كافٍ لجميع المصطلحات والاختصارات ، والمشكلات التي يمكن أن تواجه المستخدم أثناء عملية الاستخدام.

4. إمكانية مشاركة البيانات **(Data Sharing Capability) :**

يجب أن تكون قاعدة البيانات الخاصة بالأداة قابلة للعمل عليها من خلال شبكة الحاسب بحيث يتشارك فيها جميع المستخدمين بكفاءة للوصول إلى البيانات أو إدخال بيانات جديدة من خلال نظام أمني **(Security System)** محكم توزع من خلاله الصلاحيات **(Privileges)** على المستخدمين.

5. قابلية التحديث **(Upgrade Acceptability) :**

يجب أن يتم معرفة العديد من العوامل التي تتعلق بتحديث الأداة من حيث :

- قابلية تحديث نماذج التقدير (Estimation Models) المستخدمة داخل الأداة ، وذلك من قبل المستخدم من خلال وسيلة سهلة.
- تعهد المورد بعمل تحديثات مستمرة للأداة لكي تدعم وظائف جديدة ومنصات تشغيل جديدة ونماذج تقدير جديدة ، وذلك لزيادة كفاءتها.
- الثمن الذي سوف تدفعه الشركة أو المؤسسة مقابل هذه التحديثات.

6. الدعم الفني **(Technical Support)**

رغم أن أدوات التقدير البرمجية أصبحت في متناول الجميع وسهلة الاستخدام إلا أنها تستخدم نماذج رياضية معقدة ربما تجعل المستخدم يحتاج الإجابة على بعض التساؤلات والدعم الفني ، لذا يجب معرفة :

- هل المورد لديه الإمكانيات اللازمة للرد على أية استفسارات بسرعة ومن خلال وسائل متعددة؟ مثل : الدعم الفني عن بُعد.
- هل يقدم المورد دورات تدريبية للعملاء لزيادة كفاءتهم في استخدام الأداة؟.
- هل يعقد حلقات توعية للعملاء لإطلاعهم على كل ما هو جديد في مجال تقدير تكلفة المشاريع البرمجية وليس فقط ما يتعلق باستخدام أداته البرمجية؟.

7. كيفية تحديد حجم وأفق المشروع **(How Project Scope and Size is Specified) :**

يجب أن تتسم الأداة البرمجية بالمرونة الكافية في تحديد حجم وأفق المشروع من خلال وجود خيارات متعددة ، لذا يجب معرفة هذه الخيارات من خلال الإجابة على هذه التساؤلات :

- ما هي الخيارات المتاحة لإدخال حجم وأفق المشروع؟. ومن أهم تلك الخيارات : عدد سطور الشفيرة **(Line Of Code "LOC")** ونقاط الدالة **(Function Points)** ومكونات واجهة المستخدم الرسومية **(Graphical User Interface "GUI" Components)** ، عدد البرامج الفرعية **"Subprograms"**(مثل الأصناف **"Classes"** والطرق **"Methods"** ، والوحدات المركبة **"Modules"** ، الدوال **"Functions"**).
- كيف يتم إدخال بيانات حجم المشروع؟ هل تتم بإدخال قيمة واحدة للحجم **(Size Single Value)** (مثل : **LOC = 150000 or FP = 130**) ، أو يتم إدخاله كمدى من الأعداد (مثل : **Expected Value = 150000 , Low value = 120000 , High Value = 180000**) ، أم يتم إدخاله كعدد للوحدات البرمجية المركبة **(Modules)** ، ..... وخلافه.

8. نماذج التقدير المدعومة **(Estimation Models Supported) :**

يجب أن تكون النماذج المدعومة من قبل الأداة البرمجية من النماذج المشهود لها بالثقة والتي تم تقويمها من قبل العديد من الخبراء المهتمين بمجال تقدير تكلفة المشاريع البرمجية ، ومع ذلك وبغض النظر عن ما يمكن أن تتعلمه وتعرفه حول الخوارزميات الداخلية المستخدمة في أداة التقدير ، ما يجب أن تحدده هو : هل استخدام هذه الأداة في تقدير تكلفة المشاريع البرمجية التي تقوم شركتك أو مؤسستك بتطويرها مفيد أم لا؟. للإجابة على هذا السؤال وبثقة يجب أن تحصل على نسخة تجريبية **(Evaluation or Demo Copy)** من هذه الأداة البرمجية

وتستخدمها في تقدير تكلفة مشاريع برمجية قامت شركتك أو مؤسستك بتطويرها سابقاً ، ومن ثم مقارنة نتائج التقدير الناتجة من هذه الأداة بنتائج التقدير المعروفة والمسجلة لديك لهذه المشاريع والموثوق بدقتها ، فإذا كانت النتائج متقاربة فهذا يعني إنه يمكن استخدام هذه الأداة أما إذا كانت النتائج متباعدة فهذا يعني أن استخدام هذه الأداة غير مفيد.

ومن جهة أخرى يجب تحديد هل الأداة البرمجية (وبالتالي النموذج المستخدم) تسمح باستخدام البيانات التاريخية **(Historical Information)** للمشاريع السابقة ، وكيفية تنفيذ ذلك؟ ؛ حيث إن بعض الأدوات يتم معايرتها **(Calibrated)** بطريقة يدوية (أي يجب أن تقوم باستنتاج قيم عوامل المعايرة بنفسك من خلال البيانات التاريخية لديك) ، والبعض الآخر من الأدوات تتم المعايرة بطريقة آلية (أي إنك تقوم بإدخال قياسات المشروع مثل : الحجم الفعلي والمجهود والجدول الزمني ومن ثم تقوم الأداة بتعديل عوامل النموذج).

ويجب أيضا ، معرفة حجم المشروع (أقل حجم وأقصى حجم) يمكن للأداة البرمجية أن تستخدمه في التنبؤ الدقيق لتكلفة المشروع ؛ حيث إنه يوجد العديد من نماذج التقدير **(Estimation Models)** لديها حدود **(Limitations)** أو قيود على حجم المشروع الذي يمكن االتنبؤ بتكلفته بصورة دقيقة. فعلى سبيل المثال نموذج **COCOMOII** لا يمكن معايرته للمشاريع ذات حجم أقل من 2000 سطر (2000 >SLoc) ، لذا يجب ألا تستخدم المشاريع ذات الأحجام الصغيرة أدوات التقدير التجارية في تقدير التكلفة والمجهود.

9. موجهات التكلفة الأخرى **(Other Cost Drivers)** :

بصفة عامة ، تسمح لك أدوات التقدير البرمجية بتعيين عدد من مسببات التكلفة أو الإنتاجية **(Cost Or Productivity Drivers)** (مثل : إمكانيات فريق العمل وخبرتهم ومتطلبات دورة حياة المشروع واستخدام الأدوات ، ....وخلافه) ،

ومع ذلك يجب معرفة كل المسببات التي تتيحها لك الأداة لاستخدامها لتمثيل مسببات التكلفة أو الإنتاجية ، وهل هي تتناسب وتكفي ما تحتاجه شركتك أو مؤسستك عند تقدير تكلفة المشاريع الخاصة بها أم لا؟.

10. القيود والأولويات **(Constraints and Priorities) :**

يجب معرفة ما هي القيود (مثل : أقصى فترة زمنية لتطوير المشروع ، وأقصى تكلفة ممكنة وأقصى عدد من فريق التطوير ، وخلافه من القيود) والأولويات (مثل : الأولوية لأقل مجهود وبالتالي لأقل عدد من فريق التطوير ، الأولوية لأقصر فترة زمنية يمكن تطوير المشروع خلالها ، الأولوية لأقل تكلفة ممكنة وخلافه من الأولويات) التي تتيحها لك الأداة عند استخدامها لتقدير التكلفة ، وهل هي بالفعل تتيح لك ما تحتاجه أم لا؟.

11. المخرجات **(Outputs) :**

تُعد المخرجات المطبوعة على الأوراق **(Hardcopy)**أو المخرجات المرنة المخزنة على وسائط التخزين المختلفة **(Softcopy)** من أهم السمات التي يجب أن تتصف بها أداة التقدير البرمجية ؛ حيث إن مثل هذه المخرجات (التقارير) تساعد على مناقشة خيارات التقدير المختلفة بوضوح مع العملاء ، لذا يجب الحرص على معرفة نوعية التقارير التي تقوم بتوليدها الأداة وتحديد مدى الإفادة منها في عملية التقدير ، وكذلك معرفة تسمح الأداة بنسخ هذه التقارير على وسائط تخزين خارجية لاستخدامها فيما بعد عن الحاجة أم لا؟.

12. إمكانية تصدير واستيراد البيانات **(Data Import / Export Capability) :**

تُعد هذه السمة من السمات المهمة للإفادة من خبرات المشاريع السابقة من خلال استيراد بياناتها التاريخية لمعايرة الأداة البرمجية المستخدمة لتحديث النماذج الرياضية الموجودة بها ، وكذلك تصدير نتائج التقدير إلى برمجيات أخرى للإفادة من إمكانياتها المختلفة مثل : برنامج **MS Projects** ، وبرنامج **MS Excel** .

أسئلة تقويم ذاتي

1.ما هي أهم السمات والشروط التي يجب أن تؤخذ بعين الاعتبار عند تقويم أدوات تقدير تكلفة المشاريع البرمجية؟.

2. ضع علامة (√) أمام الإجابة الصحيحة وعلامة (x) أمام الإجابة الخطأ. ثم صحح الإجابة الخطأ

ا) نماذج التقدير الخوارزمية عبارة عن دوال رياضية مستنبطة إما على أساس نظري أو على أساس تجريبي ولا حاجة على الإطلاق للبديهة والخبرات.

ب) لا يوجد نموذج تقدير خوارزمي ملائم لشتى أنواع البرمجيات وجميع بيئات التطوير البرمجي.

ج) نموذج بوتنام يمكن أن يوصف بدالة جاوشين (Gaussian Distribution).

د) قيمة الثابت "C" لنموذج بوتنام ثابت لحالات التخطيط المختلفة.

هـ) يعد نموذج كوكومو أشهر نماذج تقدير تكلفة المشاريع البرمجية في الوقت الحالي لأنه مفتوح وموثق جيدا ومتاح للجميع ولكن يعيبه التعقيد بعض الشيئ.

و) تعرف الشيفرة الجديدة في نموذج كوكوكو II بأنها الشيفرة المكتوبة بطريقة يدوية بدون استخدام أي من الأدوات البرمجية.

ز) ليس هناك أدوات تقدير برمجية بدقة مطلقة لحل مشكلة تقدير تكلفة المشاريع البرمجية ، ولكنها في نفس الوقت تعد وسيلة مفيدة ضمن طاقم الأدوات الخاص بعملية التقدير.

نشاط

وعليك الآن عزيزي الدارس أن تبحث في شبكة الإنترنت عن أهم أدوات تقدير تكلفة المشروعات البرمجية ، والتي سوف نجد من بينها :

- Construx Estimate , USC COCOMO II , COSTAR

ثم حاول التعرف عليها والمقارنة بينها طبقاً لما سبق شرحه ، واختر أحدها وحاول فهمه واستخدامه في عملية تقدير تكلفة بعض من المشاريع البرمجية.

الخلاصة

اشتملت هذه الوحدة على

- نماذج الخوارزمية لتقدير البرمجيات : الشكل العام لمعظم هذه النماذج :

$$E = A^{*}(size)^{B} * EMF$$

- مقياس الصناعة لدقة تنبؤ نموذج التكلفة وهو يحسب قيمة الخطأ النسبي من خلال المعادلة:

$$MRE = \left\langle \frac{(Epred - \text{Eact})}{\text{Eact}} \right\rangle$$

–تقويم نماذج تقدير التكلفة معايير تقوم نماذج تقدير التكلفة التعريف والدقة و الموضوعية والبناء الاستدلالي والتفاصيل والاستقرار والأفق وسهولة الاستخدام والتوقع المستقبلي والتدبير.

– أنواع النماذج الخوارزمية:

* نموذج بونتام :

الشكل العالم للنموذج يكون طبقاً المعادلة :ـــ $M(t) = 2Kate^{-at^2}$

حيث M(t) عدد الأشخاص ، في أي وقت "t" ، K الجهد الكلي ، a يمثل عامل التعجيل.

– نموذج Putnam : يعتمد على الإنتاجية : ويأخذ هذا النموذج المعادلة:

$$E = \mathrm{B}^{*}(s/\mathrm{p})^{3} * 1/T_{\mathrm{d}}{}^{4}$$

"E" الجهد اللازم لتطوير البرمجية ، "T_d" مدة تطوير المشروع مقدرة بالشهور أو السنين ، "B" عامل الخبرات الخاصة ، "p" عامل الإنتاجية.

– نموذج بوهم (كوكومو): أشهر النماذج لتقدير التكلفة يستخدم المعادلتين:

$$E = \mathrm{a} * \mathrm{Size}^{\mathrm{b}}$$

$$TDEV = 2.5 * \mathrm{E}^{\mathrm{c}}$$

حيث "E" جهد التطوير ،" Size" حجم البرمجية ، "TDEV" الوقت اللازم لتقدير البرمجية ، "a" ، "b" ، "c" عوامل تعتمد على شكل التطوير.

– نموذج كوكومو الثاني : يتميز عن النموذج السابق الموديل السابق يتطلب معرفة مبدئية لحجم البرمجية كأساس في حين أن هذا الموديل يأخذ جهود تخمين مختلفة – والموديل السابق يعطي تقديراً للجهد والوقت في حين أن هذا الموديل يعطي مدى للتقدير الذي يقع الجهد المطلوب في حدوده والموديل الحالي يتكيف مع متطلبات إعادة الاستخدام والهندسة العكسية في حين أن الموديل السابق لا يأخذ في الاعتبار

هذه العوامل. ويتضمن نموذج COCOM II أربعة نماذج فرعية: نموذج تكوين التطبيق ونموذج التصميم المبكر ونموذج إعادة الاستخدام والنموذج المعماري اللاحق.

– أدوات تقدير تكلفة المشروعات البرمجية : يمكن أن تكون منتجات برمجية بمفردها أو ضمن منتجات برمجية فعالة ، هناك العديد من السمات والشروط التي يجب أن تؤخذ عند تقويم تقدير تكلفة المشاريع البرمجية: السعر ومنصة التشغيل والإدارة وسهولة الاستخدام والتوثيق وإمكانية مشاركة البيانات وقابلية التحديث والدعم الفني وكيفية تحديد حجم وأفق المشروع ونماذج التقدير المدعومة وموجهات التكلفة الأخرى والقيود والأولويات والمخرجات وإمكانية تصدير واستيراد البيانات.

لمحة مسبقة عن الوحدة التالية

في الوحدة التالية سنتناول خطوات تقديرتكلفة المشروعات البرمجية ، وفيها نناقش خطوات تقدير تكلفة المشروعات البرمجية ، حيث يتم استعراض الغرض من كل خطوة ، والخطوات التنفيذية لتحقيق الغرض منها ، والدخل والخرج الخاص بها ، والمسئولين القائمين على تنفيذها.

إجابة التدريبات

تدريب (1)

الجهد الكلي "K" يمكن حسابه بالتعويض بالقيم المعطاة أعلاه في المعادلة رقم (1) ، ومنها تحصل على K = 7812.5 Person-Years ، وحيث إن جهد التطوير يساوي 40% من الجهد الكلي : إذن E = 0.4*7812.5 = 3082 Person-Years

تدريب (2)

$E = 3*(32^{1.12}) = 145.51$ Person-Month (PM)
$TDEV = 2.5 * (145.51)^{0.35} = 14.3$ Month
Productivity = Size/E = 32000/145.51 = 220 LOC/PM
Average Stuffing = E/TDEV = 145.51/14.3 = 10.2 Persons

تدريب (3)

EAF = CPLX * STOR * AEXP * PCAP * DATA
= 1.15 * 1.06 * 1.13 * 1.17 * 1.16 = 1.87
$E = 3*(32^{1.12})$ * 1.87 = 272.1 Person-Month (PM)
$TDEV = 2.5 * (272.1)^{0.35} = 17.79$ Month
Productivity = Size/E = 32000/272.1 = 117.6 LOC/PM
Average Stuffing = E/TDEV = 272.1/17.79 = 15.3 Persons

تدريب (4)

عدد نقاط الدالة = NOP =2*10 + 5*8 + 10*6 = 120 **نقطة دالة**

تقدر الإنتاجية الخاصة ببيئة ICASE بالمرتفعة (High) ، وتقدر إنتاجية فريق التطوير بالمنخفضة (Low) ، وبالتالي يمكن أن تقدر الإنتاجية الكلية بمتوسط الإنتاجيتين وهو إنتاجية عادية (Nominal) ، وبالتالي فهي تساوي 13.

إذن الجهد = 120*(1 – 0.25) / 13 = 90 / 13 = 7 شخص – شهور.

تدريب (5)

- **عدد نقاط الدالة = NOP =2*10+3*5 +5*8 + 10*12 = 195 نقطة دالة.**
- **عدد أسطر الشيفرة = 195 * 53 = 10335 سطر**
- **عوامل الأس (Exponent Scale Factors) :**

PREC = Low = 4.96 ، FLEX = Very High = 1.01 ، RESL = Very Low = 7.07 ، TEAM = Nominal = 3.29 ، PMAT = Nominal = 4.68

- **موجهات التكلفة :**

RCPX = Relay + DATA + CPLEX + DOUC = Very High + Nominal + Low + Nominal = 5 + 3 + 2 + 3 = 13 = High = 1.33

PDIF = TIME + STORE + PVOL = Nominal + High + Nominal = 3 + 4 + 3 = 10 = High = 1.29

FCIL = TOOL + SITE = Low + Nominal = 2 + 3 = 5 = Low = 1.1

PERS = PREX = SCED = REUSE = Nominal = 1

$$E = a * Size^{b} * EAF$$

$$b = 0.91 + 0.01 \sum_{i=1}^{i=5} SF_i$$

b = 0.91 + 0.01 (4.96+1.01+7.07+3.29+4.68) = 1.1201

$$EAF = \prod_{i=1}^{i=7} EM_i$$

EAF = 1.33*1.29*1.1*1*1*1*1 = 1.887

$E = 2.94 * (10.335)^{1.1201} * 1.887 = 75.9$ Person-Months

تدريب (6)

- الجهد المطلوب لدمج الشيفرة المولدة آلياً = E_{Auto}

E_{Auto} = ASLOC * (AT/100) / ATPROD = (8000*45/100)/2400 = 1.5 Person-Months .

- **الجهد المطلوب لتعديل ودمج باقي الشيفرة المعادة للاستخدام = E_{equiv}**
- **عدد أسطر الشيفرة المكافئة بالكليو سطر = KEASLOC**

KESLOC = KASLOC * (1 – AT/100) * AAM
AAM = (AA + SU + 0.4DM +0.3CM + 0.3IM) / 100 =
(5 + 20 + 0.4 *40 + 0.3*60 + 0.3*80)/100 = 0.83
KESLOC = 8 * (1- 0.45) * 0.83 = 3.652
$E = 2.94 * (10.335 + 3.652)^{1.1201} * 1.887 = 106.52$ Person-Months
E_{equiv} = 106.52 – 75.9 = 30.62 Person-Months
الجهد الإضافي المطلوب = $E_{Auto} + E_{Equiv}$ = 32.12 شخص – شهور.

مسرد المصطلحات

مرونة التطوير "Development Flexibility "FLEX" :

وهو مقياس يعكس درجة المرونة في عملية التطوير.

دقة التصميم المعماري/تحليل المخاطر " (Architecture/Risk Resolution RESL ") :

وهو مقياس يعكس درجة تحليل المخاطر التي تمت أثناء التصميم المعماري للمنتج البرمجي.

تماسك وتناغم الفريق "Team Cohesion "TEAM :

وهو مقياس يعكس درجة تماسك الفريق من حيث معرفتهم لبعضهم البعض ، ومدى خبرتهم في العمل كفريق متكامل ، ومدى توافق أهدافهم وثقافتهم ولغاتهم، ومدى القدرة والرغبة في تعاونهم وتفاعلهم مع بعضهم البعض .... وخلافه.

كمال العملية "Process Maturity "PMAT :

وهو مقياس يعكس مدى نضج واكتمال عمليات المؤسسة.

الشيفرة الجديدة (New Code) في نموذج COCOMO II :

الشيفرة المكتوبة بطريقة يدوية بدون استخدام أي من الأدوات البرمجية (CASE Tools).

الشيفرة معادة الاستخدام Reused Code :

وهي شيفرة موجودة مسبقاً (Pre-Existed Code) وتم استخدامها كما هي بدون أي تغيير وبدون محاولة لفهمها ، ويطلق عليها أيضاً شيفرة الصندوق الأسود (Black Box Code) ، والجهد المطلوب في هذه الحالة يساوي الصفر.

الشيفرة المعاد تهيئتها Adapted Code :

وهي شيفرة موجودة مسبقاً وتم تعديلها لتدمج مع باقي المكونات البرمجية للمنتج البرمجي ، ويطلق عليها أيضاً شيفرة الصندوق الأبيض (White Box Code) ، وفي هذه الحالة يوجد جهد لفهم الشيفرة قبل تعديلها وجهد لتعديلها حتى تعمل بنجاح مع باقي المكونات البرمجية.

المصطلح بالإنجليزية	معناه بالعربية
Advanced	متقدم
Algorithmic Estimations Models	نماذج التقدير الخوارزمية
Application Composition Model	نموذج تكوين التطبيق
" Architecture/Risk Resolution " RESL	دقة التصميم المعماري/تحليل المخاطر
Automated Software Tools	الأدوات البرمجية المؤتمتة
Automatically Translated Code	الشيفرة المعاد تحويلها آلياً
Average Stuffing	متوسط عدد فريق العمل
Black Box Code	شيفرة الصندوق الأسود
COCOMO Model	ونموذج Boehm
Cohesion Team	تفاهم فريق التطوير
"COTS" Commercial off-the-shelf Components	المكونات التجارية الجاهزة
Constraints and Priorities	القيود والأوليات
Constructiveness	البناء الاستدلالي
Data Import / Export Capability	إمكانية تصدير واستيراد البيانات
Data Sharing Capability	إمكانية مشاركة البيانات
Demo Copy	نسخة تجريبية
Developer's Experience and Capability	خبرة وإمكانيات المطور
"FLEX" Development Flexibility	مرونة التطوير
Development Process Maturity	نضج عمليات التطوير
Early Design Model	نموذج التصميم المبكر
Ease of Use	سهولة الاستخدام
Easy of Use and Documentation	سهولة الاستخدام والتوثيق
Effort Adjustment Factor	عامل ضبط الجهد
"EAF" Effort Adjustment Factor	عامل ضبط للجهد

المصطلح بالإنجليزية	معناه بالعربية
Embedded	مضَّمن
Empirical-Based Model	نماذج على أساس تجريبي
Equivalent Source Line of Code "ESLOC"	أسطر الشيفرة المكافئة
Estimation Models Supported	نماذج التقدير المدعومة
Evaluating of Cost Estimation Models	تقويم نماذج تقدير التكلفة
Excellent Rated Program	حالة التخطيط الممتاز
Exponent Term	معامل الأس
Fidelity	الدقة
Fine Tune Calibration	معايرة دقيقة
Good Rated Program	حالة التخطيط الجيد
Hardcopy	المطبوعة على الأوراق
Historical Data	بيانات تاريخية
Intermediate	متوسط
Intuition	البديهة
Linear Programming	البرمجة الخطية
Little Innovation	قليل من الابتكار
Magnitude of Relative Error	قيمة الخطأ النسبي
Mean Magnitude of Relative Error "MMRE"	متوسط قيمة الخطأ النسبي
Medium Constrain for Deadline	قيود متوسطة على وقت التسليم
Medium Innovation	ابتكار متوسط
Module	وحدة البرمجة المركبة
New Code	الشيفرة الجديدة
No Tight Deadline	قيود صارمة على وقت التسليم
Object Complexity	تعقيد الكائن

المصطلح بالإنجليزية	معناه بالعربية
Object Points	نقاط الكائن
Objectivity	الموضوعية
Organic	مترابط
Other Cost Drivers	مسببات أخرى للتكلفة
Overall Staff Distribution over Development Time	توزيع عدد فريق العمل في أي وقت خلال تطوير المشروع
Parsimony	التدبير
Percent of Integration Required for Software Adapted	النسبة المئوية للمجهود المطلوب لدمج مكونات إعادة الاستخدام
Person-Years	الجهد الكلي مقيس بشخص ــ سنة
Platform and Performance	منصة التشغيل والأداء
Poor Rated Program	حالة التخطيط الردئ
Post-Architecture Model	النموذج المعماري اللاحق
"PREC" Precedentedness	سابقة الأعمال
Privileges	الصلاحيات
"PMAT" Process Maturity	كمال العملية
Productivity	الإنتاجية
Prospectiveness	التوقع المستقبلي
Model (Putnam (SLIM	نموذج Putnam
Re-Engineering	الهندسة العكسية
Regression Analysis	التحليل الانحداري
Requirements Evolution and Volatility	تطور وعدم ثبات المتطلبات
Reusable Components	المكونات المعادة للاستخدام
Reuse Model	نموذج إعادة الاستخدام
Risk Management Techniques	منهجيات إدارة المخاطر

المصطلح بالإنجليزية	معناه بالعربية
Scope	الأفق
Self-Descriptiveness	الوصف الذاتي
Semi-Detached	شبه مترابط
Softcopy	نسخ على وسائط التخزين المختلفة
Software Projects Cost Estimation Tools	أدوات تقدير تكلفة المشروعات البرمجية
Stability	الاستقرار
Stable Development Environment	بيئة تطوير مستقرة
Standard Deviation	انحراف معياري
Statistical Simulation	المحاكاة الإحصائية
Subjective	موضوعية
System Level	مستوى النظام
"TEAM" Team Cohesion	تماسك وتناغم الفريق
Technical Support	الدعم الفني
Theory–Based Models	نماذج على أساس نظري
Types of Algorithmic Models	أنواع النماذج الخوارزمية
Upgrade Acceptability	قابلية التحديث
What-if Analysis	ماذا إذا؟
White Box Code	شيفرة الصندوق الأبيض

المراجع

أولاً : المراجع العربية :

[1] روجر بريسمان ، "هندسة البرمجيات" ، ترجمة مركز التعريب والترجمة بالدار العربية للعلوم ، الطبعة الأولى ، 2004م.

[2] مهندس عبد الحميد بسيوني ، "أساسيات هندسة البرمجيات" ، دار الكتب العلمية للنشر والتوزيع ، القاهرة ، 2005 م.

ثانياً : المراجع الإنجليزية

[3] Ian Somerville , "Software Engineering", Addison Wesley, 2001.

[4] Ronald J. Leach, "Introduction to Software Engineering", CRC Press, 1999.

[5] Douglas Bell , "Software Engineering : A Programming Approach", 3rd Edition, Addison Wesley.

[6] Shari Pfleeger, "Software Engineering - Theory and Practice", 2nd Edition.

[7] B. W. Boehm, Software Engineering Economics, Englewood Cliffs, NJ: Prentice-Hall, 1981.

[8] Steven C. McConnel , "Estimation", Chapter 8, Rapid Development, Microsoft Press, 1996 , www.construx.com/stevemcc

[9] Boehm, et al , “Cost Models for Future Life Cycle Processes: COCOMO 2.0”, 1995.

[10] Karen Lum and et. al , "Handbook for Software Cost Estimation" , Jet Propulsion Laboratory Pasadena, California.

WWW.ceh.nasa.gov/downloadfiles/ Web%20Links/cost_hb_public-6-5.pdf

[11] Reifer, D. , Boehm, B., and Chulani, S., "The Rosetia Stone: Making COCOMO81 Estimatea Work with COCOMOII, "Crosstalk : The Journal of Defense Software Engineering", February 1999.

[12] Albrecht, A. J., "Measuring Application Development Productivity", Proc. IBM Application Development Symposium. Monterey, CA, October 1979.

[13] Jones, Capers. Applied Software Measurement: Assuring Productivity and Quality, New York: McGraw-Hill, 1991.

[14] Albrecht, A. J., and J. E. Gaffiney, " Software Function, Source Line of Code and Development Effort Prediction : A Software Science Validation", IEEE Trans. Software Engineering, November 1983.

[15] **U.S. Air Force's Software Technology Support Center , " Chapter 13 Software Estimation, Measurement, and Metrics" http://www.stsc.hill.af.mil/resources/tech_docs/gsam3/chap13.pdf**

[16] **Conte, S. D., Dunsmore, H. E., and Shen, V. Y., Software Engineering Metrics and Models, Benjamin/Cummings Publishing Co., Inc., Menlo Park, CA 1986.**

[17] **Guillermo Sebastian Donatti , "Software Development Effort Estimations Through Neural Networks" , Faculty of Mathematics , Astronomy and Physics , Cordoba National University ,Cordoba, May 2005. http://www.freewebtown.com/sdeetnn/web/docs/sdeetnn.pdf**

[18] **Putnam, L., and W. Myers, "Measure for Excellence ", Yourdon Press, 1992.**

[19] **"Modern Empirical Cost and Schedule Estimation Tools",**

عنوان مقال منشور على الموقع :

http://www.dacs.dtic.mil/techs/estimation/comparison.shtml

[20] **Martin Glinz Arun Mukhija , "COCOMO (Constructive Cost Model)" , Seminar on Software Cost Estimation , WS 2002 / 2003 , Presented by Nancy Merlo – Schett , Requirements Engineering Research Group , Department of Computer Science , University of Zurich, Switzerland.**

[21] **Danfeng Hong. "Software Cost Estimation" , Department of Computer Science , University of Calgary , Alberta, Canada T2N 1N4. http://kdataserv.fis.fc.ul.pt/~aribeiro/cost/swCostEstimation.html**

[22] **Chapter 4 : Quality Development Costs and Schedule .**

http://www.ii.metu.edu.tr/~ion545/demo/section1.3.html

[23] **Kim Johnson , "Software Cost Estimation: Metrics and Models" , Department of Computer Science , University of Calgary , Alberta , CANADA T2N 1N4. http://sern.ucalgary.ca/courses/seng/621/W98/johnsonk/cost.htm**

[24] **Boehm, B.W., Abts, C., Clark, B., and Devna Ni-Chulani. S. (1997). COCOMO II Model Definition Manual. The University of Southern California.**

[25] COCOMO II Model Definition Manual , Version 2.1 , 1995-2000 Center For Software Engineering , USC.

[26] Hareton Leung and Zhang Fan , "Software Cost Estimation" , Dept. of Computing , Hong Kong Polytechnic University.

[27] Kathleen Peters, "Software Project Estimation", Software Productivity Center Inc.,
http://www.spc.ca/downloads/resources/estimate/estbasics.pdf

ثالثاً : مواقع على شبكة الإنترنت تم الاستفادة منها :

[28] http://sunset.usc.edu/publications/TECHRPTS/2000/usccse2000-505/usccse2000-505.pdf ,
"Software Development Cost Estimation Approaches - A Survey"

[29] www.comp.lancs.ac.uk/computing/resources/IanS/SE7/SampleChapters/ch26.pdf ,
" Software cost estimation"

[30] www.classes.cecs.ucf.edu/eel6883/ berrios/slides2/CH7-art-3-4-5.pp ,
" Software Cost Estimation "

- COCOMO II web page can be found at :
  sunset.usc.edu/research/COCOMOII/index.html
- USC COCOMO II.1999.0 Software (Implementations of Post-architecture and Early Design models) :
  http://sunset.usc.edu/research/COCOMOII/index.html

الوحدة الثامنة

خطوات تقدير تكلفة البرمجيات

محتويات الوحدة

المقدمة

تمهيد

عزيزي الدارس ،،

أهلاً بك في الوحدة الثامنة من مقرر " هندسة البرمجيات (1)" ، والتي تتناول خطوات تقدير تكلفة البرمجيات في احد عشر خطوة :

تتناول هذه الوحدة المعالم الأساسية للمشروع ، وهي الحجم والوارد وفريق التطوير والجداول الزمنية والتكلفة ، ويتم ذلك من خلال تنفيذ عدد من الخطوات التكرارية الآتية:

* تجميع وتحليل المتطلبات الوظيفية والبرمجية.
* تعريف عناصر العمل والتدابير.
* تقدير حجم المشروع البرمجي وهو من أهم العناصر التي تؤثر في تكلفة المشروع.
* تقدير مجهود تطوير المشروع البرمجي.
* الجدولية الزمنية للجهد.
* تقدير تكلفة المشروع البرمجي.
* تحديد تأثير المخاطر.
* مصادقة التقدير وتوفيقه من خلال النماذج والتناظر.
* التوفيق بين التقديرات والميزانية وفترة الجدول الزمني.
* مراجعة التقديرات والتصديق النهائي.
*متابعة التقديرات وتسجيلها والحفاظ عليها.

عزيزي الدارس أرجو أن لا يغيب عن ذهنك عند قراءة مادة الوحدة محاولة الإجابة عن الأسئلة التقويمية ، والعودة إلى النص للتأكد من صحة الإجابة ، كما أرجو عدم الرجوع إلى إجابات التدريبات إلا بعد محاولة حلها أولاً.

نأمل أن تؤدي دراستك إلى هذه الوحدة إلى إثراء فهمك وتعرفك إلى مقاييس جـــودة البرمجيات وتامينها.

أهداف الوحدة

عزيزي الدارس، بنهاية دراسة هذه الوحدة ينبغي أن تكون قادراً على أن :

- تُعرّف أدوات تقدير تكلفة المشاريع البرمجية مع ذكر أهم سماتها.
- تذكر أهم خطوات تقدير تكلفة المشاريع البرمجية مع ذكر مدخلات ومخرجات كل منها.

توطئة

بصفة عامة ، يتضمن تقدير المشروع البرمجي تقدير المعالم الأساسية للمشروع **(Project Key Milestones)** ، والحجم **(Size)** والموارد **(Resources)** ، وفريق التطوير **(Staffing)** والجداول الزمنية **(Schedules)** والتكلفة **(Cost)** ، ويتم ذلك من خلال تنفيذ عدد من الخطوات التكرارية **(Iterative Steps)** ، بسبب أن عملية التقدير نفسها تُعد جزء رئيساً من عملية التخطيط والتصميم والتي يتم خلالها تصميم المنتج البرمجي ليتطابق الأداء **(Performance)** ، والسعر **(Cost)** ، وقيود الجدول الزمني **(Schedule Constraints)** ، بجانب إعادة التوفيق **بين (Reconciliation)** مختلف التقديرات **ومراجعتها**. وسوف نتناول الآن أهم تلك الخطوات وهي كما يلي [10] :

1. الخطوة الأولى: تجميع وتحليل المتطلبات الوظيفية والبرمجية

Gathering and Analysis of Software Functional and Programmatic Requirements :

الغرض من هذه الخطوة هو جعل متطلبات المشروع البرمجي المطلوب تطويره كاملة وواضحة ، وذلك من خلال ما يلي :

- تحليل المتطلبات الوظيفية بكل دقة إلى أقل مستوى ممكن من التفاصيل ، مع توضيح المتطلبات غير المفهومة جيداً لكي تؤخذ في الاعتبار بطريقة معينة كخطر من مخاطر عملية التقدير **(Estimation Risks)** عن إجراء عملية إعادة ضبط عملية التقدير لتأخذ في الحسبان مخاطر التقدير المتوقعة.
- تحليل التدرج الهرمي للمعمارية الفيزيائية **(Physical Architecture**

(Hierarchy للمشروع البرمجي بناءً على المتطلبات الوظيفية ، بحيث يتكون من قطع برمجية **(Software Segments)** يتم تطويرها من خلال تقسيم كل منها إلى أقل مستوى ممكن من الدوال البسيطة.

- تعيين القيود البرمجية والمتطلبات التي تفرض قيوداً على عملية التطوير : مثل الميزانية **(Budget)** ، الجدول الزمني للتنفيذ **(Schedules)** ، ....وخلافه.

ويتمثل خرج هذه الخطوة فيما يلي :

- المتطلبات والقيود الفنية والبرمجية ، والفروض **(Assumptions)** الخاصة بها.
- متطلبات المشروع الوظيفية المحسنة والطرق المستخدمة في عملية التحسين.
- التدرج الهرمي لمعمارية المشروع كقطع برمجية مقسمة إلى أقل مستوى ممكن من الدوال البسيطة.

ويُعد مديرو البرامج **(Software Manager)** ، ومهندسو النظم **(System Engineers)** ، والمهندسون المعنيون بالمشروع **(Cognizant)** هم المسئولين عن تنفيذ مهام هذه الخطوة.

2. الخطوة الثانية : تعريف عناصر العمل والتدابير

Definition of Work Elements and Procurements :

الغرض من هذه الخطوة هو تعريف عناصر العمل **(Work Elements)** وقائمة التدابير **(Procurements List)** اللازمة لتطوير المشروع البرمجي والتي سوف تُضمن في عملية التقدير ، وذلك من خلال :

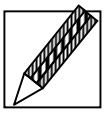
- تعيين فئات هيكلية تجزئة العمل **(Work Breakdown Structure "WBS" Categories)** والتي تتضمن النشطات المختلفة عبر دورة حياة المشروع البرمجي بدءاً من تحليل المتطلبات حتى إتمام اختبار المشروع ، وهي نموذجياً كالتالي:

- فئة إدارة المشروع البرمجي **(Software Project Management)**.
- فئة هندسة نظم البرمجيات **(Software Systems Engineering)**.
- فئة هندسة البرمجيات **(Software Engineering)**.
- فئة هندسة تكامل واختبار البرمجيات **(Software Integration and Test Engineering)**.
- فئة بيئة تطوير البرمجيات **(Software Development Environment)**.
- فئة ضمان جودة البرمجيات **(Software Quality Assurance)**.
- فئة صيانة ودعم البرمجيات **(Software Maintenance and Support)**.

- تعيين خصائص عناصر العمل التي سوف تقود إلى تكلفة في الحجم والمجهود ، وكذلك تعيين الخصائص التي يمكن أن تقود إلى مخاطر محتملة ، ومن ثم تعيين قائمة بتلك الخصائص والمخاطر، ومن أهم الأمثلة على هذه الخصائص ما يلي :
  - إعتمادية **(Reliability)** النظام المطلوبة.
  - حجم قاعدة البيانات **(Database)** المستخدمة.
  - مدى تعقيد الوحدات البرمجية المركبة **(Complexity of Modules)** داخل النظام.
  - مدى ونسبة إمكانية استخدام بعض المكونات البرمجية القابلة لإعادة الاستخدام **(Reusable Components)**.
- التوثيق **(Documentation)** الإضافي المطلوب.
  - كل ما هو جديد ولم يتم استخدامه وتجربته من قبل مثل : أسلوب البرمجة ولغتها و طريقة التصميم والتشفير.
  - مدى استخدام أدوات التطوير **(Development Tools)** والخبرة بها خلال مراحل دورة حياة المشروع.
  - المخاطر المصاحبة لاستخدام البرمجيات الجاهزة من طرف ثالث **(Third Party Ready Software)** التي سوف تكون جزء لا يتجزأ من المشروع.

- مستوى قدرة وخبرة ومهارة فريق تطـوير المشـروع مثـل المحللـين "**Analysts**" ، المبرمجين "**Programmers**") على استخدام اللغـة والأدوات التي سوف يطور بها المشروع.
- احتمالية عدم استمرار بعض من فريق التطوير ذات الخبرة طوال فتـرة تطوير المشروع.
- المتطلبات الغامضة غير المكتملة
- التطوير المتزامن للأجهزة والمعدات اللازمة لتشغيل المشروع (**Hardware**).
- قيود زمن التنفيذ (**Execution Time Constraints**).
- ضـغط الجـدول الزمنـي للتطـوير (**Development Schedule Compression**).
- قيود الذاكرة المطلوبة (**Memory Constraints**).

تدريب (1)



عين قائمة التـدابير(**Procurements List**) اللازمـة لتطـوير مشـروع برمجي

ويتمثل خرج هذه الخطوة في :

- قائمة بعناصر العمل التي سوف تقود إلى التكلفة والفروض الخاصة بها.
- قائمة بالتدابير اللازمة لتطوير المشروع البرمجي.
- قائمة بالمخاطر المحتملة.

ويُعد مديرو البرامج (**Software Manager**) ، ومهندسـو الـنظم (**System Engineers**) ، والمهندسون المعنيون بالمشروع (**Cognizant**) هم المسئولين عـن تنفيذ مهام هذه الخطوة.

3. الخطوة الثالثة : تقدير حجم المشروع البرمجي

Estimation of Software Project Size :

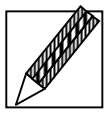
يُعد حجم المشروع البرمجي أهم العناصر التي تؤثر في تكلفته، وكما أوضحنا سابقاً فإن تقدير التكلفة يصبح أكثر دقة كلما تقدمت عملية التطوير ، ويمكن معرفته على وجه الدقة عندما ينتهي العمل به ، ولكن عملية التقدير يجب أن تتم قبل التنفيذ الفعلي للمشروع ، لذا يجب إجراء التحليل اللازم لتعيين اتجاه تغير حجم المشروع البرمجي **(Software Project Size Trend)** مع تقدم مراحل تطويره المختلفة ، وذلك من خلال العوامل التالية في الاعتبار :

- عدد المتطلبات الوظيفية **(Functional Requirements)** المختلفة المذكورة في وثيقة مواصفات متطلبات البرمجية **(Software Requirement Specification "SRS")** ، وثيقة مواصفات متطلبات الواجهة **(Interface Requirement Specification "IRS").**
- عدد وحدات البرمجية الموجودة في خطة تطوير البرمجية **(Software Development Plan "SDP")** ، أو في وثيقة وصف تصميم البرمجية **(Software Design Description "SDD").**
- تقدير عدد سطور شفيرة المصدر **(Source Lines-of-Code "SLOC")** ، أو عدد نقاط الوظيفة **(Function Points "FP")** لكل بند من بنود تكوين البرمجية **(Software Configuration Items "SCI").**

وحيث إن حجم المشروع البرمجي يؤثر تأثيراً مباشراً على تكلفة التطوير الكلية ، وكذلك على الجدول الزمني لعملية التطوير ، فإن أي انحراف مؤثر **(Significant Deviation)** في تقديره سوف يؤدي إلى ظهور المشكلات التالية أثناء عملية التطوير :

- ترجمة غير واقعية لتقدير أصل المتطلبات والموارد المطلوبة لتطوير البرمجية.
- تقدير خطأ للمجهود **(Faulty Effort Estimation)** المطلوب لتطوير البرمجية.
- مشكلات في النماذج **(Models)** والأساس المنطقي **(Rationale)** المستخدمة في عملية التقدير.
- مشكلات فيمدى استقرار المتطلبات **(Requirements Stability)** ، التصميم **(Design)** ، التشفير البرمجي **(Coding)** ، وفي معظم عمليات التطوير بصفة عامة.

تدريب (2)



يُعد تقدير حجم المشروع البرمجي أول تحدي جدي يقابل مخطط المشروع علل لذلك

وبصفة عامة يوجد أربع طرق لتقدير حجم البرمجية **(Software Size)** ، تم تناولها في الوحدة السادسة من هذا المقرر، وحيث إن معظم نماذج تقدير التكلفة التجارية **(Commercial Cost Models)** تستخدم مقياس سطور الشيفرة المنطقية لتقدير التكلفة ، يمكنك اتباع ما يلي لتقدير **LOC** :

1. استخدم خصائص عناصر العمل التي سوف تقود إلى تكلفة في الحجم والمجهود **(Cost Drivers Elements)** ، والتي تم تعينها في الخطوة السابقة ، وكذلك تصنيف المتطلبات الوظيفية للبرمجية **(Software Functional Requirements)** إلى الفئات التالية من إرث البرمجية **(Software Heritage)** :

- تصميم جديد وشيفرة جديدة **(New Design and New Code)**.
- تصميم مشابه وشيفرة جديدة **(Similar Design and New Code)**.
- تصميم مشابه مع إعادة استخدام بعض الشيفرات **(Similar Design and**

.Some Code Reuse)

- تصميم مشابه مع إعادة استخدام معظم الشـيفرات (Similar Design and Extensive Code Reuse).

2. قم بتقدير الحجم البرمجي لكل وظيفة برمجية كالتالي :

- في حالة إعادة استخدام الوظيفة البرمجيـة أو تعـديلها (Reusable , or Modifiable) فيمكن لمدير المشـروع أن يسـتخدم التنـاظر مـع حكـم الخبـراء (Analog with Expert Judgment) ، أو التنـاظر مـع بيانـات تاريخيـة (Analog with Historical Data) لوظائف مشابهة في أنظمة أخرى لتقدير الحجم البرمجي لهذه الوظيفة البرمجية.
- في حالة تطوير شيفرة مشابهة أو جديدة تماماً استخدم طريقة بيـرت الإحصـائية (Statistical PERT(Program Evaluation and Review Technique) Approach) ، حيث تكون الخبرات السابقة والبيانات التاريخية شبه محدودة ، ويمكن تلخـيص هـذه الطريقة في التالي :

- خمن تقديراً مبدئياً (Initial Best Guess) ، ويفضل أن يكون بالتنـاظر بقـدر الإمكان ، ومن ثم افترض أن هذا التقدير هو أقل حجم ممكن (Least).
- استخدم الخبرات في تقدير أقصى حجـم ممكـن (Most) ، مـع مراعـاة أن يكون الفرق بين "Least" و "Most" كبيـر فـي حالـة كـون متطلبـات الوظيفة غير مفهومة أو غير واضحة.
- استخدم الخبرات والبيانات التاريخية بقدر الإمكـان لتقـدير الحجـم الأقـرب للحقيقة (Likely).
- احسب الحجم المتوقع (Mean) من المعادلة التالية :

$$\text{Mean} = (\text{Least} + 4*\text{Likely} + \text{Most}) / 6$$

- استخدام الجدول رقم 8.1 لتصحيح المجهود الناتج من استخدام التقدير النـاتج LOC طبقاً لفئة أرث البرمجية.

جدول رقم 8.1 : فئات إرث البرمجية ومضروب الجهد المقابل لها [10]
Effort Multipliers of Software Heritage Categories

مضروب الجهد **Effort Multiplier**	فئة إرث البرمجية **Software Heritage Category**
1.2	**تصميم جديد وشيفرة جديدة** **New Design and New Code**
1.0	**تصميم مشابه وشيفرة جديدة** **Similar Design and New Code**
0.8	**تصميم مشابه وجزء من شيفرة معاد الاستخدام** **Similar Design and New Code**
0.6	**تصميم مشابه ومعظم الشيفرة معادة الاستخدام** **Similar Design and New Code**

3. في حالة اعتماد تقديرات الحجم على قواعد بيانات تاريخية ، أو التناظر مع مشاريع سابقة ، مستخدمة مقياس عدد أسطر الشيفرة الفيزيائية ، قم بتحويل هذا المقياس إلى مقياس عدد أسطر الشيفرة المنطقية طبقاً للجدول رقم 6.2 الموضح بالوحدة السادسة من هذا المقرر.

4. قم بتجميع تقديرات الحجم لمختلف وظائف البرمجية للحصول على التقدير النهائي والإجمالي لحجم البرمجية.

مخرجات الخطوة الثالثة هي :

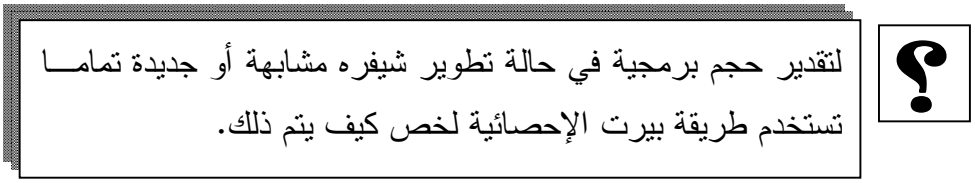
1. الفرضيات **(Assumptions)** التي تم أخذها في الاعتبار أثناء تقدير حجم البرمجية.

2. الطرق التي تم استخدامها في تقدير حجم البرمجية.

3. تقديرات حجم البرمجية لكل دالة (وظيفة) بعدد أسطر الشيفرة المنطقية **(LOC)** ، والفئة البرمجية الموروثة لكل دالة (وظيفة).

4. تقدير حجم البرمجية الكلي ممثلاً في عدد أسطر الشيفرة المنطقية **(LOC)**.

أسئلة تقويم ذاتي



لتقدير حجم برمجية في حالة تطوير شيفره مشابهة أو جديدة تماما تستخدم طريقة بيرت الإحصائية لخص كيف يتم ذلك.

4. الخطوة الرابعة:تقدير مجهود تطوير المشروع البرمجي

Estimation of Software Project Development Effort :

الهدف الأساسي من هذه الخطوة هو : تحويل حجم البرمجية المقدر في الخطوة السابقة إلى مجهود لتطوير البرمجية ، وهنا سوف نورد بعض الاقتراحات التي يمكن إتباعها لحساب المجهود اللازم لتطوير برمجية ما.

- الاقتراح الأول :

1. حساب المجهود اللازم لتطوير كل عنصر "وظيفة" من عناصر"وظائف" العمل **(Work "Functions" Elements)** التي تم تحديدها في الخطو ة الأولى باستخدام المعادلة التالية :

SW_Development_Effort = SW_Size_Estimate / SW_Development_Productivity

حيث :

- **SW_Development_Effort** هو : جهد تطوير البرمجية (شخص ــ شهور "PM").
- **SW_Size_Estimate** هو : حجم البرمجية المقدر (LOC).
- **SW_Development_Productivity** هو : إنتاجية المبرمج **(LOC/PM)**.

2. استخدم التقدير المبني على التشابه **(Estimation by Analogy)** لحساب المجهود مباشرة ، وفي حالة عدم توافر بيانات تاريخية **(Historical Data)**

لمشاريع مماثلة ، قم بالبحث عن بيانات موثقة يمكن الحصول منها على إنتاجية المبرمج بشرط توافق عملية التطوير **(Development Process)** المبني عليها هذه البيانات وتلك المستخدمة في عملية تطوير المشروع الحالي، فمثلاً في حالة استخدام عملية التطوير التزايدي **(Incremental Development Process)** يمكن اعتبار متوسط الإنتاجية (325 – 244) [10].

3. استخدام الجدول رقم 8.1 لتصحيح المجهود الناتج من استخدام التقدير الناتج **LOC** طبقاً لفئة أرث البرمجية.

4. قم بتجميع تقديرات المجهود لمختلف وظائف البرمجية للحصول على التقدير الإجمالي للمجهود المطلوب لتطوير البرمجية البرمجية، هذا التقدير الإجمالي يتضمن فقط جهد تطوير البرمجية (النشاطات المصاحبة لتطوير البرمجية مثل : هندسة نظم البرمجيات وهندسة البرمجيات وهندسة اختبار البرمجيات) ، و لا تتضمن باقي الجهد اللازم لباقي عناصر العمل المسجلة في هيكلية عناصر العمل (WBS) والتي تم الحصول عليها في الخطوة الأولى مثل جهد الإدارة **(Management Effort)** وجهد ضمان جودة البرمجية **(Quality Assurance Efforts)** .

5. في حالة الرغبة في تقسيم الجهد المبذول لتطوير البرمجية المحسوب في البند (4) إلى مستويات أدنى من التفصيل يمكن استخدام الجدول رقم 8.2.

6. استخدم الجدول رقم 8.3 لحساب الجهد اللازم لباقي عناصر العمل المسجلة في هيكلية عناصر العمل **(WBS)** والتي تم الحصول عليها في الخطوة الأولى.

جدول رقم 8.2 : تحليل فئات هيكلية عناصر عمل تطوير البرمجيات [10]
Decomposition of Software Development WBS Categories

فئة هيكلية عناصر العمل **WBS Category**	النسبة المئوية لمتوسط جهد التطوير **% Mean of Development Effort**
هندسة نظم البرمجيات **SW System Engineering**	%15
هندسة البرمجيات **SW Engineering**	%63
هندسة اختبار البرمجيات **SW Test Engineering**	%22
إجمالي تطوير البرمجية	%100

7. قم بتجميع تقديرات الجهد لمختلف عناصر العمل الموجود بالجدول السابق ، ومن ثم إضفها إلى الجهد الناتج في البند (4) للحصول على تقدير الجهد النهائي اللازم لتطوير البرمجية.

- الاقتراح الثاني :

هو استخدام أداة (برمجية) من أدوات (برمجيات) تقدير تكلفة المشاريع البرمجية موثوق بها وتدعم عملية تحويل الحجم إلى جهد ، وجدول زمني ، وتكلفة بطريقة مباشرة. وإن شاء الله سوف نتناول أهم خصائص تلك الأدوات لاحقاً في هذا الفصل.

- الاقتراح الثالث :

هو استخدام منهجية خوارزمية **(An Algorithmic Approach)** ، مثل نموذج كوكومو **(Boehm's Cocomo Model)** ، لتحويل حجم البرمجية ممثلة في عدد الأسطر **(LOC)** إلى جهد.

جدول رقم 8.3 : النسب المئوية لجهود الأنشطة الإضافية لجهد تطوير البرمجية[10]

% Additional Activities Efforts to Software Development Effort

فئة هيكلية عناصر العمل WBS Category	النسبة المئوية لمتوسط جهد التطوير % Mean of Development Effort
إدارة البرمجيات Software Management	أضف : 6-27%
دعم الاختبار على مستوى النظام System-Level Test Support includes SW Development Test-Bed	أضف : 34-112%
ضمان جودة البرمجيات Software Quality Assurance	أضف : 6-11%
التحقق من تكاملية النظام والمصادقة عليها IV&V	أضف : 9-45%
إدارة تكوين المشروع Project Configuration Management	أضف : 3-6%
إدارة المشروع Project Management	أضف 8-11 %
إدارة الفوز بصفقة المشروع Acquisition Management	أضف : 11-22%
إعادة العمل Re-Work	أضف 17-22%
صيانة الخمس سنوات الأولى Maintenance First Five Years	أضف 22% من جهد تطوير البرمجية سنوياً

مخرجات الخطوة الرابعة هي :

1. الفرضيات **(Assumptions)** التي تم أخذها في الاعتبار أثناء تقدير الجهد المبذول لتطوير البرمجية.
2. الطرق التي تم استخدامها في تقدير الجهد.
3. تقديرات جهد البرمجية لكل دالة (وظيفة) ، مع الأخذ في الاعتبار موروثة الدالة (الوظيفة).
4. تقدير الجهد اللازم لعناصر العمل غير المرتبطة بتطوير البرمجية ارتباطاً مباشراً.
5. الجهد الإجمالي اللازم لتطوير البرمجية.

5. الخطوة الخامسة : الجدولة الزمنية للجهد

: (Schedule the Effort)

الغرض من هذه الخطوة هو تقدير الفترة الزمنية اللازمة لإتمام تطوير جميع مراحل المشروع البرمجي ، أي تقدير الفترة الزمنية التي سيتم خلالها جدولة تنفيذ جميع المجهودات اللازمة لتطوير جميع عناصر العمل التي تم تسجيلها في الخطوة الثانية **(Elements of the WBS)** ، وذلك تحت قيود معينة مثل الحجم ، والمجهود ، والتكلفة ، وفريق العمل ، والترتيب الذي سيتم من خلاله تطوير هذه العناصر ، وكذلك اعتمادية كل عنصر على العنصر الآخر من حيث أولوية التنفيذ.

وبصفة عامة ، يوجد العديد من الخيارات لحساب الفترة الزمنية بدلالة المجهود **(Effort)** ، نذكر منها ما يلي :

1. **الخيار الأول :** استخدام أداة تقدير **(Estimation Tool)** لحساب الفترة اللازمة التي سيتم خلالها جدولة الجهد زمنياً **(Schedule the Effort)** عن طريق إدخال حجم البرمجية والجهد إلى أداة التقدير.
2. **الخيار الثاني :** استخدام البيانات التاريخية **(Historical Data)** الخاصة بالمشاريع السابقة المماثلة والتي تم تطويرها عن طريق شركتك أو مؤسستك.

3. **الخيار الثالث :** استخدام إحدى نماذج التقدير التجريبية البسيطة ، والتي من أهمها المعادلة : $S = A * E^B$ ، حيث **"E"** تمثل الجهد المطلوب لتطوير البرمجية مقيسة بشخص ــ شهور **(Person-Months)** ، و**"S"** الفترة الزمنية الكلية اللازمة لجدولة هذا الجهد **(Schedule's Total Amount of Time)** مقيسة بالشهر **(Month)** ، و **"A"** ، و **"B"** عوامل تحدد من خلال التحليل الانحداري **(Regression Analysis)** مستخدماً البيانات التاريخية لمشاريع مماثلة سابقة ، وفي حالة عدم وجود مثل هذه البيانات ، ومن خلال المنشورات العديدة في هذا الصدد ، وجد أن A=3 ، B=1/3 تُعد بداية طيبة لحساب الفترة الزمنية الكلية اللازمة لجدولة الجهد.

4. **الخيار الرابع :** استخدام التقديرات التقريبية **(Ballpark Schedule Estimates)** ، **وذلك** من خلال النظر في مجموعة من الجداول المعدة من قبل خبراء قاموا بتحليل العديد من المشاريع التي تم تنفيذها ، ويُعد هذا الخيار جيد في حالة عدم توافر بيانات تاريخية لمشاريع مماثلة للمشروع الذي تقوم بتطويره. (أنظر المرجع رقم [8] لمزيد من المعلومات حول هذا الخيار).

السؤال الآن : ماذا بعد إتمام خطوة حساب فترة الجدولة الزمنية **(Schedule)** ؟.
في حالة حساب فترة الجدولة الزمنية لكل مرحلة من مراحل تطوير المشروع ، يجب أن تكون النتائج متوافقة إلى حد ما مع الخبرة التاريخية **(Historical Experience)** في هذا المجال ، والمدونة بالجدول رقم 8.4 ، وكذلك في حالة حساب فترة الجدولة الزمنية للمشروع للكل ، يمكن استخدام البيانات المدونة في الجدول نفسه لتقسيمها على مراحل المشروع المختلفة.

جدول رقم 8.4: تخصيص وقت التنفيذ على مراحل تطوير البرمجية[10]

Allocation of Schedule Time Over Software Development Phases

النسبة المئوية لمتوسط جهد التطوير **% Mean of Development Effort** (بيانات تاريخية)	مراحل تطوير البرمجية **Software Phases**
18	**تحليل المتطلبات** **Requirements Analysis**
22	**تصميم البرمجية** **Software Design**
36	**التنفيذ** **Implementation**
24	**تكامل واختبار البرمجية** **Software Integration and Test**

مخرجات الخطوة الخامسة هي :

1. الفرضيات **(Assumptions)** التي تم أخذها في الاعتبار أثناء تقدير فترة الجدولة الزمنية لتطوير المشروع.
2. الطرق التي تم استخدامها في تقدير فترة الجدولة الزمنية لتطوير المشروع.
3. تقدير فترة الجدولة الزمنية اللازمة لتطوير عناصر العمل.
4. فترة الجدولة الزمنية اللازمة لتطوير كامل البرمجية.

6. الخطوة السادسة : تقدير تكلفة المشروع البرمجي

Software Project Cost Estimation :

الغرض من هذه الخطوة هو : تقدير التكلفة الكلية للمشروع البرمجي بحيث تغطي جميع مجهودات عناصر العمل والتدابير اللازمة لإتمام المشروع.

والآن يمكنك اتباع ما يلي لتقدير التكلفة الكلية للمشروع :

1. تحديد تكلفة التدابير **(Determine The Cost of Procurements)** وهي كما يلي :
   - تحديد تكلفة الخدمات والدعم ، مثل محطات العمل **(Workstations)** ، اللوحات الخاصة بالاختبارات **(Test-Bed Boards)** والمحاكيات **(Simulators)** ، معدات الدعم **(Support Equipments)** وشبكات الاتصال والحاسب والهواتف.
   - تحديد تكلفة التدابير البرمجية مثل نظم التشغيل **(Operating Systems)** والمترجمات **(Compilers)** والتراخيص **(Licenses)** ، وأدوات التطوير **(Development Tools)** .
   - تحديد تكلفة السفريات والرحلات **(Travel and Trips)** المتعلقة بمراجعة العميل **(Customer Reviews)** زويارة البائعين **(Vendor Visits)** ، وحضور المؤتمرات المتعلقة بالمشروع **(Project-Related Conferences)**.
2. تحديد تكلفة التدريب المخطط للمشروع البرمجي.
3. تحديد المرتب الشهري ومستوى مهارة فريق العمل.
4. إدخال الجهد **(Effort)** ، ومستويات المرتبات **(Salary Levels)** ، تكلفة التدابير إلى أداة برمجية مدعومة من المنشأة لحساب التكلفة الكلية ، وباقي العوامل الخارجية الأخرى التي قد تؤثر في التكلفة مثل الضرائب التي سوف

تدفعها المنشأة في حالة تنفيذها المشروع لحساب جهة أخرى ، وكذلك معدلات التضخم وزيادة الأسعار في حالة امتداد تنفيذ المشروع خلال أكثر من سنة.

5. في حالة وجود تضارب **(Inconsistencies)** أو ثغرات أثناء حساب التكلفة ، وهذه حقيقة يمكن أن تحدث في حالة التوفيق بين هذه التكلفة والميزانية المخصصة للمشروع ، ربما يكون من الضروري تكرار التقدير مرات متعددة للخطوات الأخرى المتعلقة بالتكلفة وتعديل العوامل ذات التأثير المباشر في التكلفة مثل تقليل الجهد والتدابير ، أو فرض مخاطر زيادة في حالة عدم كفاية المفترض منها للتوافق مع الميزانية المفروضة للمشروع البرمجي ، وسوف نناقش ــ إن شاء الله ــ هذا الموضوع بالتفصيل في الخطوة التاسعة.

مخرجات الخطوة السادسة هي :

1. الفرضيات **(Assumptions)** التي تم أخذها في الاعتبار أثناء تقدير التكلفة الكلية لتطوير المشروع.
2. الطرق التي تم استخدامها في تقدير التكلفة الكلية لتطوير المشروع.
3. تكلفة التدابير بالعملة المحلية أو بالدولار.
4. تكلفة كل عنصر من عناصر العمل المسجلة في **WBS** بالعملة المحلية أو بالدولار.
5. التكلفة الكلية بالعملة المحلية أو بالدولار.

7. الخطوة السابعة : تحديد تأثير المخاطر

Determine the Impact of Risks:

الغرض من هذه الخطوة هو تحديد : تأثير المخاطر التي يمكن أن يتعرض لها المشروع على التكلفة الكلية وإعادة تقديرها بناءً على ذلك، ولتحقيق هذا الغرض يمكنك اتباع الخطوات التالية :

1) اطلع على قائمة المخاطر الابتدائية التي تم تحديدها في الخطوة الثانية ، ومن ثم اختر أهم المخاطر التي يمكن أن تؤثر تأثيراً سلبياً كبيراً على تطوير المشروع.
2) قم بتقدير تأثير هذه المخاطر كلا على حده على التكلفة الكلية للمشروع التي تم حسابها في الخطوة السابقة ، ويمكنك الاستعانة بالبيانات المدونة بالجدول رقم 8.5 لتحديد موجهات المخاطر **(Risk Drivers)** ، ونسبة تأثيرها على التكلفة الكلية.

جدول رقم 8.5 : موجهات المخاطر ونسبة تأثيرها على التكلفة الكلية[10]

Cost Drivers and it's Effects on the Total Cost

العوامل التي تقود إلى المخاطر (Risk Drivers)	التأثير على التكلفة المقدرة		
	عالي High	عالي جداً Very High	عالي زيادة Extra High
2. الخبرة وفريق العمل (Experience and Teaming) : • خبرة محدودة بالمشاريع البرمجية. • لم يشارك فريق التطوير في المراحل الأولية لتخطيط وتصميم المشروع. • لا يوجد تكامل بين فريق البرمجيات وفريق المعدات.	1.02	1.05	1.08
2. التخطيط (Planning) :	1.10	1.17	1.25

العوامل التي تقود إلى المخاطر (Risk Drivers)	التأثير على التكلفة المقدرة		
	عالي High	عالي جداً Very High	عالي زيادة Extra High
• عدم وجود تخطيط مفصل بالدرجة المناسبة ، مع مراجعة غير دقيقة وغير كافية. • عدم مشاركة جميع الشركاء في تخطيط عملية التطوير. • افتراض فرضيات متفائلة غير مدروسة وخصوصاً في تحديد موروثية البرمجيات.			
2. المتطلبات والتصميم (Requirements & Design) : • عدم الانتهاء من معمارية النظام والبرمجيات في وقت مبكر مناسب ، مع عدم وضوح التقسيم الوظيفي للمعدات والبرمجيات. • أخذ قرارات نظامية مهمة بدون حساب تأثيرها على البرمجيات ، وبالتالي على المشروع. • توقع ثبات المتطلبات في مرحلة متأخرة من دورة حياة المشروع.	1.05	1.13	1.20
2. فريق التطوير (Staffing) : • توقع عالي لمغادرة أعضاء من فريق التطوير المشروع قبل انتهائه. • مشاركة فريق التطوير في مرحلة متأخرة من مراحل تطوير المشروع. • التخطيط لتسريح فريق العمل قبل إتمام عمليات تجميع واختبار وتسليم المشروع.	1.02	1.05	1.13
2. الاختبار (Test) : • عدم وجود محاكيات كافية لاختبار المشروع. • عدم وجود فريق اختبار مخصص ، والتخطيط لمشاركة فريق التطوير في عمليات الاختبار في مرحلة متأخرة من دورة حياة	1.05	1.08	1.15

العوامل التي تقود إلى المخاطر (Risk Drivers)	التأثير على التكلفة المقدرة		
	عالي High	عالي جداً Very High	عالي زيادة Extra High
المشروع. • تطوير خطة الصيانة في مرحلة متأخرة من دورة حياة المشروع.			
2. الأدوات (Tools) : • عدم وجود أو وجود محدود لأدوات إدارة التكوين والاختبارات. • مشاركة فريق التطوير في مرحلة متأخرة من مراحل تطوير المشروع. • اختيار أدوات للتصميم غير موثوق بها ولوقت محدد للتحليل.	1.02	1.03	1.10
أعلى تأثير متوقع على التكلفة = حاصل ضرب التأثيرات المدونة لكل عامل.	1.30	1.6	2.32

3) مع افتراض أن تأثير كل عامل لا يؤثر على العامل الآخر ، قم بضرب تأثيرات العوامل المختلفة المختارة في بعضها للحصول على عامل التأثير الكلي (Total Impact Factor) ، ومن ثم قم بضبط تقدير التكلفة المحسوب في الخطوة السابقة بضربه في عامل التأثير الكلي للحصول على التقدير الكلي لتكلفة المشروع متضمناً المخاطر التي يمكن أن تواجه أثناء تطويره.

مخرجات الخطوة السابعة هي :

1. الفرضيات (Assumptions) التي تم فرضها أثناء تعديل التكلفة لتتضمن تأثير المخاطر.
2. الطرق التي تم استخدامها في تعديل التكلفة لتتضمن تأثير المخاطر.
3. قائمة مفصلة بالمخاطر المتوقع أن يواجهها المشروع البرمجي أثناء عملية التطوير.

4. القيم المعدلة لكل من : الحجم **(Size)** ، والجهد **(Effort)** ، والجدول الزمني **(Schedule)** ، والتكلفة **(Cost)** ، وذلك نتيجة تضمين المخاطر في عملية التقدير.

8. الخطوة الثامنة : مصادقة وتوفيق التقدير من خلال النماذج والتناظر:

Validate and Reconcile the Estimation via Models and Analogy :

الغرض من هذه الخطوة هو : التثبت من صحة التقدير من خلال مقارنته بالتقديرات التي ستنتج من استخدام طرق أخرى مثل النماذج التجريبية ، والتناظر مع مشاريع أخرى مماثلة. ولتحقيق هذا الغرض اتبع الخطوات التالية :

1. قم بإجراء تقدير ثاني ، بالإضافة إلى التقدير السابق الذي تم الحصول عليه في الخطوة السابقة ، مستخدماً إحدى الطرق التالية :

- تكليف مثمن أو فريق تثمين آخر على نفس مستوى خبرة المثمن أو فريق التثمين الذي قام بالتقدير الأول لإجراء تقدير ثاني مستقل.
- مقارنة التقديرات الحالية مع التقديرات الخاصة بالمشاريع الأخرى السابقة والمماثلة مستخدماً البيانات التاريخية **(Historical Data)** لهذه المشاريع ، فعلى سبيل المثال يمكنك أن تقارن :
  - الحجم والمجهود والتكلفة لمشاريع مماثلة.
  - الحجم مقابل الدوال (مقارنة حجم الدوال المماثلة).
  - الحجم مقابل المجهود والتكلفة (مقارنة إنتاجية التطوير لأحجام مماثلة).
  - تقنية التطوير مقابل المجهود والتكلفة (مقارنة إنتاجية التطوير لتقنيات تطوير مماثلة)
  - مستخدماً إحدى النماذج التجريبية.

2. قم بعقد اجتماع للمسئولين عن التقدير الأول والتقدير الثاني لمناقشة أهم الاختلافات بين التقديرين وحل هذه الخلافات والوصول إلى تقدير متفق عليه من الجميع.

مخرجات الخطوة الثامنة هي :

1. الفرضيات **(Assumptions)** التي تم فرضها أثناء التثبت من التقديرات.
2. الطرق التي تم استخدامها للتثبت من التقديرات.
3. القيم المعدلة والمصادق عليه لكل من : الحجم **(Size)** ، والجهد **(Effort)** ، والجدول الزمني **(Schedule)** ، والتكلفة **(Cost)**.

9. الخطوة التاسعة التوفيق بين التقديرات والميزانية وفترة الجدول الزمني

Reconcile Estimates , Budget, and Schedule :

الغرض من هذه الخطوة هو : مراجعة التقديرات المصادق عليها من قبل فريق التثمين في الخطوة السابقة وتقدير مدى توافقها مع القيود المفروضة على تنفيذ المشروع مثل الميزانية وفترة الجدول الزمني ، والتوفيق بينهم.

ولتحقيق الغرض من هذه الخطوة يمكنك إتباع الخطوات التالية :

1. احسب هامش الميزانية **(Budget Margin)** من المعادلة التالية :

Budget Margin = (Estimated Cost–Imposed Budget Cost)/ Imposed Budget Cost * 100

2. احسب هامش فترة الجدولة الزمنية **(Schedule Margin)** من المعادلة التالية :

Schedule Margin = (Estimated Schedule –Imposed Schedule)/ Imposed Schedule * 100

3. قارن التكلفة المقدرة **(Estimated Cost)** ، وفترة الجدولة الزمنية المقدرة **(Estimated Schedule)** ، وهوامشهما مع التكلفة وفترة الجدولة الزمنية وهوامشهما المفروضة على تنفيذ المشروع ، وحدد مدى التوافق أو الاختلاف بينها.

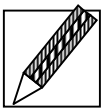
4. في حالة ما إذا كانت التقديرات أعلى بكثير ، قم بتحديد وحل الاختلافات كما يلي :
   a. أعد تنقيح الأفق المطلوب للمشروع والوظائف الخاصة به حتى تصل إلى أقل مستوى ممكن ، وذلك بتحليل أفق ووظائف المشروع وتعيين أولوياتها لتحديد الوظائف الممكن تجاهلها ، مع الأخذ في الاعتبار العلاقات الداخلية بين جميع الوظائف ، وقم بإزالة الوظائف التي يمكن تجاهلها وما يترتب على ذلك من هيكلية تقسيم المشروع **(WBS)**.
   b. ابدأ بإزالة التدابير غير الضرورية بصفة مطلقة.
   c. قم بإعادة تقدير فترة الجدولة الزمنية ، والتكلفة ، والمخاطر لتضمين الخفض الذي حدث في a, b في التقديرات المعدلة.
   d. قم بتكرار a, b, c حتى يتم التوافق بين التقديرات المعدلة والقيود المفروضة على المشروع من ميزانية وفترة جدولة زمنية.
5. راجع وظائف المشروع والتدابير التي تم إقرارها بعد تخفيضها ، وكذلك التقديرات المعدلة مع راعي المشروع للوصول إلى اتفاق.
6. قم بتعديل هيكلية تقسيم العمل **(WBS)** بناءً على الوظائف المعدلة **(Revised Functionality)**.

مخرجات الخطوة التاسعة هي :

1. الفرضيات **(Assumptions)** التي تم فرضها أثناء تعديل التقديرات.
2. الطرق التي تم استخدامها في تعديل التقديرات.
3. القيم المعدلة لكل من الحجم **(Size)** ، والجهد **(Effort)** ، والجدول الزمني **(Schedule)** ، والتكلفة **(Cost)** ، وذلك نتيجة إجراء عملية التوافق بين التقديرات والقيود المفروضة على تنفيذ المشروع من ميزانية وفترة جدولة زمنية.
4. قائمة بالوظائف والتدابير المعدلة.
5. هيكلية تقسيم العمل المعدلة.

6. عامل تأثير المخاطر المعدل ، وذلك بعد تعديل المخاطر طبقاً لتعديل وظـــائف المشروع.

تدريب (3)



تُعد خطوة التوفيق بين التقديرات والميزانية وفترة الجدول الزمنـــي من أصعب الخطوات في عملية التقدير وضح ذلك

10. الخطوة العاشرة : مراجعة التقديرات والتصديق النهائي عليها

Review and Approve the Estimates :

الغرض من هذه الخطوة هو مراجعة تقديرات المشروع البرمجي والحصول على موافقة إدارة المشروع ، حيث يتم التصديق على هيكليـــة تقســـيم المشـــروع **(WBS)** ، وهيكله البرمجي **(Software Architecture)** ، وكذلك المصادقة علـــى التقديرات بعد التأكد من خلوها من الأخطاء والمشـــاكل ، والتحقـــق مـــن الطـــرق المستخدمة في استنتاج الحجم ، والجهد ، وفترة الجدولة الزمنية ، والتكلفـــة، وفـــي هذه المرحلة ربما يكون من الضروري توقيع عقد اتفاق.

مخرجات الخطوة العاشرة هي :

1. المشكلات الموجودة في التقديرات.
2. القيم المعدلة والمراجعة والمصادق عليها لكل من الحجـــم **(Size)** ، والجهـــد **(Effort)** ، والجدول الزمني **(Schedule)** ، والتكلفة **(Cost)** .
3. اتفاق عمل (إذا كان ضرورياً).

أسئلة تقويم ذاتي

1.في أي خطوة يتم التصديق على هيكلية تقسيم المشروع وهيكله البرمجية ، وكذلك المصادقة على التقديرات بعد التأكد من خلوها من الأخطاء والمشاكل والتحقق من الطرق المستخدمة في استنتاج الحجم ، والجهد، وفترة الجدولة الزمنية؟

2. ضع علامة (√) أمام الإجابة الصحيحة وعلامة(X)أمام الإجابة الخطأ. ثم صحح الإجابة الخطأ؟

ا) يمكن زيادة دقة تقدير حجم المشروع البرمجي بتفكيك المشروع إلى وحدات صغيرة وتقدير حجم كل وحده على حده ومن ثم تجميع هذه النتائج للحصول على الحجم الكلي للمشروع.

ب) لتحسين عملية التقدير والتخطيط داخل منشأتك في المستقبل، قم بتوثيق وأرشفة جميع البيانات الحقيقية لعمليات التقدير التي يتم تعديلها والتصديق عليها في كل مرحلة من مراحل دورة حياة المنتج.

ج) نموذج كوكومو عبارة عن نموذج رياضي يستخدم لتحويل حجم البرمجية ممثلة في عدد من السطور إلى جهد.

د) خيار استخدام التقديرات التقريبية لحساب الفترة الزمنية بدلالة المجهود يعتبر خياراً جيداً في حالة توافر بيانات تاريخية لمشاريع مماثلة للمشروع الذي يتم تطويره.

11.الخطوة الحادية عشر متابعة التقديرات والحفاظ عليها

Track , Report , and Maintain the Estimates :

الغرض من هذه الخطوة هو التأكد من دقة التقديرات مع مرور الوقت ، وتسجيل البيانات الخاصة بها باعتبارها وثائق يتم الاحتفاظ بها لاستخدامها بيانات تاريخية لإجراء تقديرات لمشاريع مماثلة في المستقبل. ولتحقيق هذا الغرض اتبع الخطوات التالية :

1. تتبع التقديرات لتحديد : متى **(When)** ، وبأي قدر **(How Much)** ، ولماذا **(Why)** ربما تتجه تقديرات المشروع البرمجي إلى معدلات أعلى **(Over Running)** أو معدلات منخفضة **(Under Running)** من المقدر لها ، وقارن تلك التقديرات بمثيلاتها لمشاريع سابقة لتحديد مدى تغيرها مع مرور وقت التطوير ، وذلك للتعرف على مدى جودة التقديرات التي تمت وكيفية تغييرها على مدى دورة حياة المشروع البرمجي.
2. قم بتوثيق الفروق بين التقديرات الحالية والتقديرات الخاصة بالمشاريع المماثلة السابقة.
3. ولتحسين عملية التقدير والتخطيط داخل منشأتك في المستقبل ، قم بتوثيق وأرشفة جميع البيانات الحقيقة لعمليات التقدير التي يتم تعديلها والتصديق عليها في كل مرحلة من مراحل دورة حياة المنتج البرمجي أو عند كل معلم أساسي من معالم **(Milestones)** المشروع ، ومن أهم البيانات التي يمكن توثيقها وأرشفتها ما يلي :

- المعلومات الأساسية ذات الصلة بالمشروع **(Project Related Main Information)** : مثل :

- اسم المشروع (Project Name).
- منصة التشغيل (Platform).
- لغة التطوير (Development Language).
- طرق التقديرات والفرضيات (Estimation Methods and Assumptions).
- تاريخ التصديق على التقديرات (Date of Approved Estimates) .

- القيم التقديرية (Estimated Values) والقيم الواقعية (Real Values) لكل من : الحجم ، والجهد ، وتكلفة الجهد ، وتكلفة التدابير (Cost of Procurements) لكل عنصر من عناصر العمل المسجلة في هيكلية تقسيم المشروع (WBS) ، وكذلك القيمة الإجمالية لكل منها.
- تواريخ المخطط الزمني الواقعية والمقدرة لكل معلم أساسي من معالم (Milestones) المشروع.
- المخاطر التي تم أخذها في الاعتبار وتأثيراتها الواقعية والمقدرة على سير تطوير المشروع.

مخرجات الخطوة الحادية عشر وهي :

1. المقارنات بين البيانات الحقيقة والمقدرة الناتجة عن تتبع سير المشروع ، وتقويمها.
2. القيم المُحدَّثة (Updated Values) لكل من الحجم (Size) ، والجهد (Effort) ، والجدول الزمني (Schedule) ، والتكلفة (Cost) ، والمخاطر وتأثيرها.
3. توثيق وأرشفة جميع بيانات المشروع متضمنة البيانات الواقعية والمقدرة (Actual and Estimated Data) .

الخلاصة

– تتناول هذه الوحدة المعالم الأساسية للمشروع ، و ذلك من خلال تنفيذ عدد من الخطوات التكرارية :

* الخطوة الأولى : تجميع وتحليل المتطلبات الوظيفية والبرمجية الغرض من هذه الخطوة جعل متطلبات المشروع البرمجي المطلوب تطويره كاملة وواضحة ، ويعد مديرو البرامج ، ومهندسو النظم ، والمهندسون المعنيون بالمشروع هم المسئولينعن تنفيذ مهام هذه الخطوة.

* الخطوة الثانية :تعريف عناصر العمل والتدابير والغرض من هذه الخطوة هو تعريف عناصر العمل وقائمة التدابير اللازمة لتطوير المشروع البرمجي والتي سوف تضمن في عملية التقدير، ويعد مديرو البرامج ، ومهندسو النظم ، والمهندسون المعنيون بالمشروع هم المسئولين عن تنفيذ مهام هذه الخطوة. .

* الخطوة الثالثة :تقدير حجم المشروع البرمجي وهو من أهم العناصر التي تؤثر في تكلفة المشروع.

* الخطوة الرابعة: تقدير مجهود تطوير المشروع البرمجي ، والهدف الأساسي من هذه الخطوة هو تحويل حجم البرمجية المقدرة في الخطوة السابقة إلى مجهود لتطوير البرمجية.

* الخطوة الخامسة : الجدولية الزمنية للجهد والغرض من هذه الخطوة هو تقدير الفترة الزمنية اللازمة لإتمام تطوير جميع مراحل المشروع البرمجي.

* الخطوة السادسة: تقدير تكلفة المشروع البرمجي والغرض من هذه الخطوة هو تقدير التكلفة الكلية للمشروع البرمجي بحيث تغطي جميع مجهودات عناصر العمل والتدابير اللازمة لإتمام المشروع.

* الخطوة السابعة: تحديد تأثير المخاطر والغرض من هذه الخطوة هو تحديد تأثير المخاطر التي يمكن أن يتعرض لها المشروع على التكلفة الكلية وإعادة تقديرها بناء على ذلك.

* الخطوة الثامنة : مصادقة وتوفيق التقدير من خلال النماذج والتناظر والغرض من هذه الخطوة هو التثبت من صحة التقدير من خلال مقارنته بالتقديرات التي ستنتج من استخدام طرق أخرى مثل النماذج التجريبية ، والتناظر مع مشاريع أخرى مماثلة.

* الخطوة التاسعة : التوفيق بين التقديرات والميزانية وفترة الجدول الزمني والغرض من هذه الخطوة هو مراجعة التقديرات المصادق عليها من قبل فريق التثمين في الخطوة السابقة وتقدير مدى توافقها مع القيود المفروضة على تنفيذ المشروع مثل الميزانية وفترة الجدول الزمني ، والتوفيق بينهم.

* الخطوة العاشرة : مراجعة التقديرات والتصديق النهائي والغرض من هذه الخطوة هو مراجعة تقديرات المشروع البرمجي والحصول على موافقة إدارة المشروع.

*الخطوة الحادية عشر : متابعة التقديرات وتسجيلها والحفاظ عليها والغرض من هذه الخطوة هو التأكد من دقة التقديرات مع مرور الوقت ، وتسجيل البيانات الخاصة بها باعتبارها وثائق يتم الاحتفاظ بها لاستخدامها بيانات تاريخية لإجراء تقديرات لمشاريع مماثلة في المستقبل.

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس،

في الوحدة القادمة (الأخيرة) سنستعرض

"مقاييس وتأمينها جودة البرمجيات" ، وفيها نتناول تعريف الجودة وسماتها المختلفة ، والأنشطة الخاصة بتأمين جودة البرمجيات ، بالإضافة إلى شرح وافٍ لأهم مقاييس البرمجيات وتصنيفاتها المختلفة. نرجو أن تكون وحدة مفيدة لك.

إجابات التدريبات

تدريب (1)

- **مجموعة أدوات بيئة التطوير** ، مثل أدوات إدارة قاعدة البيانات **(Database Management Tools)** أودوات مراقبة النظام **(System Monitoring Tools)** أودوات توليد تقارير النظام **(Reporting Generation Tools)** وأدوات تشخيص النظام **(Diagnostic Tools)** أودوات التحليل والتصميم **(Analysis and Design Tools)**.
- **معدات ومستلزمات بيئة التطوير** ، مثل الأجهزة الخادمة **(Servers)** ومحطات العمل **(Workstations)** اولطابعات **(Printers)** وأجهزة التخزين **(Storage Devices)** ، عقود الصيانة **(Maintenance Agreements)** بويئة المحاكاة **(Simulation Environment)** ، هذا بالإضافة إلى عدد من المطورين المتزامنين **(Simultaneous Developers)**.
- **برمجيات بيئة التطوير** ، مثل : نظم التشغيل **(Operating Systems)** ، المكونات البرمجية التي سوف تصبح جزء من النظام العامل والبرمجيات المساعدة غير المضمنة في عقد التطوير مثل : أدوات الـ **CASE** ، ومترجمات اللغة **(Compilers)** ، برمجيات توليد وتحليل حالات الاختبار **(Test Case Generators and analyzers)** ، رزم البرمجيات الإحصائية **(Statistical Software Packages)**.

تدريب (2)

وذلك لارتباط مدى دقة عمليات التقدير التالية له بمدى دقة تحديده. ويُعد استخدام منهجيات متنوعة لتقدير الحجم ضرورياً للحصول على نتيجة موثوق بها ؛ حيث إن الاعتماد على منهجية واحدة يمثل المساهمة الرئيسة في المخاطر التي تنشأ

في تقدير التكلفة والجدول الزمني لتنفيذ المشروع. وأيضاً فإن دقة التقدير يمكن أن تتحسن في حالة تفكيك المشروع إلى وحدات صغيرة وتقدير حجم كل وحدة على حده ، ومن ثم تجميع هذه النتائج للحصول على الحجم الكلي للمشروع.

تدريب (3)

خطوة التوفيق بين التقديرات والميزانية وفترة الجدول الزمني تتطلب فهم المشروع البرمجي بطريقة متكاملة من حيث تكلفة كل وظيفة من الوظائف على حده ، وأهمية تنفيذها النسبية أولوية ، والعلاقات الداخلية فيما بينها.

وفي حالة وجود تضارب **(Inconsistency)** بين القيم المقدرة والقيود المفروضة فإنه من الخطأ تبرير هذا التضارب بوجود خطأ في عملية التقدير فقط ، بل يجب تنفيذ ما ينصح به الخبراء في هذا المقام ، وهو أن الحل الحقيقي يكمن في إعادة النظر في خفض الوظائف الخاصة بالمشروع دون خفض التكلفة بحذف بعض الاحتياطيات التي تم الأخذ بها أثناء عملية التقدير ، وذلك عن طريق فرض فرضيات متفائلة وغير واقعية.

مسرد المصطلحات

تعيين فئات هيكلية تجزئة العمل "Work Breakdown Structure "WBS

: Categories

تتضمن النشطات المختلفة عبر دورة حياة المشروع البرمجي بدءاً من تحليل المتطلبات حتى إتمام اختبار المشروع

الجدولة الزمنية للجهد Schedule the Effort :

هو تقدير الفترة الزمنية اللازمة لإتمام تطوير جميع مراحل المشروع البرمجي

المصطلح الإنجليزية	معناه بالعربية
Accuracy of Estimates	دقة التقديرات
Actual and Estimated Data	البيانات الواقعية والمقدرة
An Algorithmic Approach	منهجية خوارزمية
Analog with Expert Judgment	التناظر مع حكم الخبراء
Analog with Historical Data	التناظر مع بيانات تاريخية
Analysis and Design Tools	أدوات التحليل والتصميم
Assumptions	الفروض
Avionics	برمجيات تستخدم في الطيران
Ballpark Schedule Estimates	التقديرات التقريبية
Budget	الميزانية
Budget Margin	هامش الميزانية
Coding	التشفير البرمجي
Cognizant	المهندسون المعنيون بالمشروع
Commercial Cost Models	نماذج تقدير التكلفة التجارية
Complexity of Modules	تعقيد الوحدات البرمجية المركبة

المصطلح الإنجليزية	معناه بالعربية
Compressed Schedule Effort	الفترة الزمنية المضغوطة
Cost Drivers Elements	تكلفة في الحجم والمجهود
Customer Reviews	المتعلقة بمراجعة العميل
Database Management Tools	أدوات إدارة قاعدة البيانات
Date of Approved Estimates	تاريخ التصديق على التقديرات
Development Schedule Compression	ضغط جدول الزمني للتطوير
Development Tools	أدوات التطوير
Diagnostic Tools	أدوات تشخيص النظام
Documentation	التوثيق
Efficient Schedules	فترة الجدولة الزمنية الفعالة
Embedded Software	البرمجيات المُضمَّنة
Estimation by Analogy	التقدير المبني على التشابه
Estimation Risks	مخاطر عملية التقدير
Estimation Tool	أداة تقدير
Execution Time Constraints	قيود زمن التنفيذ
Faulty Effort Estimation	تقدير خطأ للمجهود
Firmware	البرمجيات الثابتة
Function Points "FP"	نقاط الوظيفة
Functional Requirements	عدد المتطلبات الوظيفية
Historical Experience	الخبرة التاريخية
Impact of Risks	تأثير المخاطر
Inconsistencies	وجود تضارب
Incremental Development Process	عملية التطوير التزايدي
Initial Best Guess	خمن تقدير مبدئي
Interface Requirement Specification "IRS"	وثيقة مواصفات متطلبات الواجهة

المصطلح الإنجليزية	معناه بالعربية
Licenses	التراخيص
Likely	الحجم الأقرب للحقيقة
Maintenance Agreements	عقود الصيانة
Memory Constraints	قيود الذاكرة المطلوبة
Models	النماذج
New Design and New Code	تصميم جديد وشيفرة جديدة
Nominal Schedules	فترة الجدولة الزمنية الظاهرية
Physical Architecture Hierarchy	التدرج الهرمي للمعمارية الفيزيائية
Process Control Software	برمجيات التحكم في العمليات
Procurements List	قائمة التدابير
Project-Related Conferences	المؤتمرات المتعلقة بالمشروع
Quality Assurance Efforts	ضمان جودة البرمجية
Rationale	الأساس المنطقي
Real-Time Software	برمجيات الوقت الحقيقي
Reconciliation	إعادة التوفيق
Reliability	اعتمادية
Reporting Generation Tools	أدوات توليد تقارير النظام
Requirements Stability	ثبات المتطلبات
Review and Approve the Estimates	مراجعة التقديرات والتصديق النهائي عليها
Revised Functionality	الوظائف المعدلة
Schedule Margin	هامش فترة الجدولة الزمنية
Shortest Possible Schedules	أقصر فترة جدولة زمنية ممكنة
Shrink-Warp Software	البرمجيات المغلفة للتوزيع التجاري
Significant Deviation	انحراف مؤثر
Similar Design and Extensive Code Reuse	تصميم مشابه مع إعادة استخدام معظم

المصطلح الإنجليزية	معناه بالعربية
	الشيفرات
Similar Design and New Code	تصميم مشابه وشيفرة جديدة
Similar Design and Some Code Reuse	تصميم مشابه مع إعادة استخدام بعض الشيفرات
Simulation Environment	بيئة المحاكاة
Simulators	المحاكيات
Simultaneous Developers	المطورين المتزامنين
Software Configuration Items "SCI"	لكل بند من بنود تكوين البرمجية
Software Design Description "SDD"	وثيقة وصف تصميم البرمجية
Software Development Environment	بيئة تطوير البرمجيات
Software Development Plan "SDP"	خطة تطوير البرمجية
Software Programmatic Requirements	المتطلبات البرمجية للبرمجية
Software Functional Requirements	المتطلبات الوظيفية للبرمجية
Software Heritage	إرث البرمجية
Software Integration and Test Engineering	هندسة تكامل واختبار البرمجيات
Software Maintenance and Support	صيانة ودعم البرمجيات
Software Manager	مدير البرامج
Software Project Management	إدارة المشروع البرمجي
Software Quality Assurance	ضمان جودة البرمجيات
Software Requirement Specification	وثيقة مواصفات متطلبات البرمجية
Software Segments	قطع برمجية
Software Size	حجم البرمجية
Source Lines-of-Code "SLOC"	عدد سطور شفيرة المصدر
Statistical PERT (Program Evaluation and Review Technique) Approach	طريقة بيرت الإحصائية
Statistical Software Packages	رزم البرمجيات الإحصائية

المصطلح الإنجليزية	معناه بالعربية
Support Equipments	معدات الدعم
SW_Development _Effort	جهد تطوير البرمجية (شخص ــ شهور "PM").
SW_Development _productivity	إنتاجية المبرمج (LOC/PM)
SW_Size _Estimate	حجم البرمجية المقدر (LOC)
System Monitoring Tools	أدوات مراقبة النظام
System Size	حجم النظام
Test Case Generators and analyzers	برمجيات توليد وتحليل حالات الاختبار
Test-Bed Boards	اللوحات الخاصة بالاختبارات
Third Party Ready Software	البرمجيات الجاهزة من طرف ثالث
Total Impact Factor	عامل التأثير الكلي
Travel and Trips	تكلفة السفريات والرحلات
Updated Values	القيم المُحدَّثة
Vendor Visits	زيارة البائعين
Work Breakdown Structure "WBS" Categories	هيكلية تجزئة العمل
Work Elements	عناصر العمل

المراجع

أولاً : المراجع العربية :

[1] روجر بريسمان ، "هندسة البرمجيات" ، ترجمة مركز التعريب والترجمة بالدار العربية للعلوم ، الطبعة الأولى ، 2004م.

[2] مهندس عبد الحميد بسيوني ، "أساسيات هندسة البرمجيات" ، دار الكتب العلمية للنشر والتوزيع ، القاهرة ، 2005 م.

ثانياً : المراجع الإنجليزية :

[3] Ian Somerville , "Software Engineering", Addison Wesley, 2001.

[4] Ronald J. Leach, "Introduction to Software Engineering", CRC Press, 1999.

[5] Douglas Bell , "Software Engineering : A Programming Approach", 3rd Edition, Addison Wesley.

[6] Shari Pfleeger, "Software Engineering - Theory and Practice", 2nd Edition.

[7] B. W. Boehm, Software Engineering Economics, Englewood Cliffs, NJ: Prentice-Hall, 1981.

[8] Steven C. McConnel , "Estimation", Chapter 8, Rapid Development, Microsoft Press, 1996 , www.construx.com/stevemcc

[9] Boehm, et al , "Cost Models for Future Life Cycle Processes: COCOMO 2.0", 1995.

[10] Karen Lum and et. al , "Handbook for Software Cost Estimation" , Jet Propulsion Laboratory Pasadena, California.

WWW.ceh.nasa.gov/downloadfiles/ Web%20Links/cost_hb_public-6-5.pdf

[11] Reifer, D. , Boehm, B., and Chulani, S., "The Rosetia Stone: Making COCOMO81 Estimatea Work with COCOMOII, "Crosstalk : The Journal of Defense Software Engineering", February 1999.

[12] Albrecht, A. J., "Measuring Application Development Productivity", Proc. IBM Application Development Symposium. Monterey, CA, October 1979.

[13] Jones, Capers. Applied Software Measurement: Assuring Productivity and Quality, New York: McGraw-Hill, 1991.

[14] Albrecht, A. J., and J. E. Gaffiney, " Software Function, Source Line of Code and Development Effort Prediction : A Software Science Validation", IEEE Trans. Software Engineering, November 1983.

[15] U.S. Air Force's Software Technology Support Center , " Chapter 13 Software Estimation, Measurement, and Metrics"

http://www.stsc.hill.af.mil/resources/tech_docs/gsam3/chap13.pdf

[16] Conte, S. D., Dunsmore, H. E., and Shen, V. Y., Software Engineering Metrics and Models, Benjamin/Cummings Publishing Co., Inc., Menlo Park, CA 1986.

[17] Guillermo Sebastian Donatti , "Software Development Effort Estimations Through Neural Networks" , Faculty of Mathematics , Astronomy and Physics , Cordoba National University ,Cordoba, May 2005.
http://www.freewebtown.com/sdeetnn/web/docs/sdeetnn.pdf

[18] Putnam, L., and W. Myers, "Measure for Excellence ", Yourdon Press, 1992.

[19] "Modern Empirical Cost and Schedule Estimation Tools",

عنوان مقال منشور على الموقع :

http://www.dacs.dtic.mil/techs/estimation/comparison.shtml

[20] Martin Glinz Arun Mukhija , "COCOMO (Constructive Cost Model)" , Seminar on Software Cost Estimation , WS 2002 / 2003 , Presented by Nancy Merlo – Schett , Requirements Engineering Research Group , Department of Computer Science , University of Zurich, Switzerland.

[21] Danfeng Hong. "Software Cost Estimation" , Department of Computer Science , University of Calgary , Alberta, Canada T2N 1N4.
http://kdataserv.fis.fc.ul.pt/~aribeiro/cost/swCostEstimation.html

[22] Chapter 4 : Quality Development Costs and Schedule .

http://www.ii.metu.edu.tr/~ion545/demo/section1.3.html

[23] Kim Johnson , "Software Cost Estimation: Metrics and Models" , Department of Computer Science , University of Calgary , Alberta , CANADA T2N 1N4.
http://sern.ucalgary.ca/courses/seng/621/W98/johnsonk/cost.htm

[24] Boehm, B.W., Abts, C., Clark, B., and Devna Ni-Chulani. S. (1997). COCOMO II Model Definition Manual. The University of Southern California.

[25] COCOMO II Model Definition Manual , Version 2.1 , 1995-2000 Center For Software Engineering , USC.

[26] Hareton Leung and Zhang Fan , "Software Cost Estimation" , Dept. of Computing , Hong Kong Polytechnic University.

[27] Kathleen Peters, "Software Project Estimation", Software Productivity Center Inc.,
http://www.spc.ca/downloads/resources/estimate/estbasics.pdf

ثالثاً : مواقع على شبكة الإنترنت تم الاستفادة منها :

[28] http://sunset.usc.edu/publications/TECHRPTS/2000/usccse2000-505/usccse2000-505.pdf ,
"Software Development Cost Estimation Approaches - A Survey"

[29] www.comp.lancs.ac.uk/computing/resources/IanS/SE7/SampleChapters/ch26.pdf ,
" Software cost estimation"

[30] www.classes.cecs.ucf.edu/eel6883/ berrios/slides2/CH7-art-3-4-5.pp ,
" Software Cost Estimation "

- COCOMO II web page can be found at :
  sunset.usc.edu/research/COCOMOII/index.html
- USC COCOMO II.1999.0 Software (Implementations of Post-architecture and Early Design models) :
  http://sunset.usc.edu/research/COCOMOII/index.html

الوحدة التاسعة

مقاييس جودة البرمجيات وتأمينها

محتويات الوحدة

المقدمة

تمهيد

عزيزي الدارس ،،

سنتعرف من خلال هذه الوحدة "مقاييس البرمجيات" وجودتها إلى وأهم السمات التي يجب أخذها في الاعتبار لإنتاج برمجيات ذات جودة عالية.

– تبدأ الوحدة بتعريف الجودة وفقاً لمقياس ايزو المعياري "B204" مع الأخذ في الاعتبار مجموعة من السمات الأساسية لإنتاج برمجيات ذات جودة عالية ، وعند الانتهاء من إنتاج البرمجية يتم التأكد من جودتها أو عدمه من خلال قياس خواص البرمجية ومراقبة وتنظيم عملية تطوير البرمجية.

– إدارة جودة البرمجيات: تشمل الأنشطة أو العمليات التي تضمن أن المشروع البرمجي يحقق أهدافه بمعنى أخر يحقق توقعات العملاء ، ويجب أن تنفصل إدارة الجودة عن إدارة المشروع لضمان الاستقلالية.

– مقاييس البرمجيات وفي هذا القسم من الوحدة نتعرف على مقاييس البرمجيات ، ونتناول خصائص مقاييس البرمجيات الجيدة ، وتصنيف مقاييس البرمجيات ، مقاييس تحديد مستوى مؤسسات تطوير البرمجيات.

عزيزي الدارس أرجو أن لا يغيب عن ذهنك عند قراءة مادة الوحدة محاولة الإجابة عن الأسئلة التقويمية ، والعودة إلى النص للتأكد من صحة الإجابة ، كما أرجو عدم الرجوع إلى إجابات التدريبات إلا بعد محاولة حلها أولاً.

نأمل أن تؤدي دراستك إلى هذه الوحدة إلى إثراء فهمك وتعرفك إلى خطوات تقدير تكلفة البرمجيات.

أهداف الوحدة

عزيزي الدارس، بنهاية دراسة هذه الوحدة ينبغي أن تكون قادراً على أن :

- تعرِّف جودة البرمجيات.
- تعدد أهم السمات الأساسية التي يجب أخذها في الاعتبار لإنتاج برمجيات ذات جودة عالية.
- تصف بإيجاز أهم السمات التي تحدد مدى جودة البرمجيات.
- تعرِّف إدارة جودة البرمجيات.
- تلخص أهم أنشطة إدارة جودة البرمجيات.
- تعدد الأسئلة الواجب الإجابة عليها أثناء التخطيط لإدارة جودة البرمجيات.
- تسرد المحتويات الواجب تضمينها في وثيقة خطط إدارة الجودة.
- تعرِّف عملية تأمين جودة البرمجيات.
- تعرِّف المقاييس المعيارية للبرمجيات مع ذكر أمثلة عليها.
- تصف الفرق بين المقاييس المعيارية والإجراءات.
- تعدد أهم أنشطة تأمين جودة البرمجيات.
- تذكر أهمية إدارة جودة البرمجيات.
- تعرِّف مقاييس جودة البرمجيات.
- تلخص في نقاط أهمية مقاييس البرمجيات.
- تذكر خصائص مقاييس البرمجيات الجيدة.
- تصنف مقاييس البرمجيات إلى أنواعها المختلفة.
- تقيس حجم البرمجية ومدى درجة تعقيدها باستخدام مقاييس ماكابي وهالستيد.
- تعدد مقاييس الجودة.
- تصنف مقاييس تحديد مستوى مؤسسات تطوير البرمجيات.

1.جودة البرمجيات (Software Quality)

أصبحت مصطلحات الجودة وتأمين الجودة للمنتجات في العالم الصناعي اليوم مهمة وشائعة الاستخدام وخاصة في صناعة البرمجيات وذلك للاعتماد المطلق على هذه الصناعة في جميع مناحي الحياة. وكحد أدنى يمكننا القول بأن برمجية ما عالية الجودة إذا ما اتسمت بالسمات العامة التالية :

- تلبي احتياجات المستخدم.
- خالية من العيوب.
- يمكن الاعتماد عليها في العمل.
- يمكن إجراء صيانة لها (تصحيح ، تغيير،.....).

وتُعرف **الجودة (Quality)** وفقاً لمقياس أيزو المعياري **"ISO Standard 8204"** بأنها : المجموع الكلي لميزات وخصائص منتج أو خدمة ما والتي تؤثر على قدرته / قدرتها على تلبية حاجات المستخدم المحددة أو الضمنية.

"The totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs"

فمثلاً نعني بجودة التصميم **(Quality of Design)** مجموعة الميزات التي يحددها المصممون لمنتج ما ، فعندما يُطلب إنتاج منتج ذي مواصفات أداء عالية ، لا مناص من زيادة جودة تصميم المنتج إذا جرى تصنيعه وفقاً للمواصفات المطلوبة. أما جودة التوافق **(Quality of Conformance)** فهي درجة اتباع مواصفات التصميم خلال مرحلة التصنيع ، وكلما ازدادت درجة التوافق ارتفع مستوى جودة التوافق ، وبالتالي جودة المنتج.

ويعرف المنتج ذو الجودة ــ أيضاً ــ بأنه المنتج الذي يلبي ويستمر في تلبية الاحتياجات التي من أجله تم إنتاجه.

والجودة لها سمة أساسية وهي أنها يمكن قياسها ، وفي هندسة البرمجيات هذه القياسات يشار لها بالمقاييس **(Metrics)** ، فعلى سبيل المثال يمكن قياس التكاليف والحجم الذي تشغله البرمجية ، وتعقيد البرمجية الخ.

ولإنتاج برمجيات ذات جودة عالية يجب الأخذ في الاعتبار مجموعة من السمات الأساسية من أهمها :

- مدى كبر أو حجم البرمجية.
- مدي تعقيد عناصر مكونات البرمجية.
- مدى الاعتماد على البرمجية.
- تكاليف إنتاج البرمجية.
- الجهد اللازم لتغيير البرمجية لتلبى احتياجات المستخدم.

حيث إن ذلك يسهم إلى حد كبير في :

- التنبؤ بجودة البرمجية خلال مرحلة تطويرها أو إنتاجها.
- قياس جودة جزء من برمجية تم إنتاجها.
- مراقبة وتحسين عملية إنتاج البرمجية.
- المقارنة بين الطرق المختلفة لإنتاج البرمجية واختيار الأفضل منها.

ويمكن الأخذ في الاعتبار سمات الجودة سابقة الذكر في واحده أو أكثر من الأوضاع التالية :

1. خارج عملية التطوير لتوضيح الأهداف.
2. خلال عملية التطوير لتوجيه عملية التطوير لتحقيق الأهداف.
3. في النهاية لتقويم إنتاج البرمجية.

ويمكن إيجاز أهم السمات التي تحدد مدى جودة البرمجيات فيما يلي:

1. الصحة **(Correctness)** : وهي تمثل المدى التي تفي فيه البرمجية بمتطلبات المستخدم.
2. الاعتمادية **(Reliability)** : وهي تمثل الحد الذي يمكن أن تعمل فيه البرمجية باستمرار دون حدوث توقف أو فشل.
3. الكفاءة **(Efficiency)** : وهي كمية RAM ووقت المعالج الذي تستخدمه البرمجية.
4. التكاملية **(Integrity)** : وهي الدرجة التي تمكن فيها البرمجية المستخدم من الدخول وتنظيم إدخال المعلومات.
5. الاستخدام **(Usability)** : سهولة استخدام البرمجية.
6. الصيانة **(Maintainability)** : **وهي** تعبر عن الجهد المطلوب لاكتشاف خطأ وإصلاحه.
7. المرونة **(Flexibility)** : وهي تعبر عن الجهد اللازم لتغير البرمجية لمقابلة التغيرات المطلوبة.
8. الاختبارية **(Testability)** : وهي تعبر عن الجهد اللازم لاختبار البرمجية بفعالية.
9. الانتقالية **(Portability)** : تعبر عن الجهد اللازم لنقل البرمجية إلى حاسبات ذات مكونات وأنظمة تشغيل مختلفة.
10. إعادة الاستخدام **(Re-usability)** : تعبر عن الحد الذي يمكن به إعادة استخدام البرمجية أو جزء منها من خلال برمجية أخرى.
11. تضافرية العمل **(Interoperability)** : تعبر عن الجهد اللازم لجعل البرمجية تعمل بالتعاون مع برمجية أخرى.

وكما هو ملاحظ فإن هناك العديد من هذه السمات متناقض مع بعضه فعلى

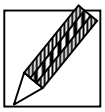
سبيل المثال لإنتاج برمجية ذات كفاءة عالية فإن إمكانية النقل أو الحمل ربما تتناقض معه ، ولهذا فإن لكل مشروع تحقيق مجموعة من الأهداف المحددة توضع في الاعتبار قبل عملية التطوير والإعداد.

ولكن عندما تنتهي من إنتاج برمجية كيف تتأكد من جودتها من عدمه؟ ، هناك طريقتان للإجابة عن هذا التساؤل :

1) قياس خواص البرمجية التي تم إنتاجها.
2) مراقبة وتنظيم عملية تطوير البرمجية.

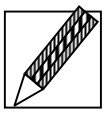
ولتوضيح الرؤيا دعنا نلاحظ أوجه الشبه بين إنتاج برمجية وإعداد وجبة غذائية من الطعام، فإذا أردت إعداد وجبة معينة فإنك تقوم بالخدمة عليها وبناءً عليه تحصل على وجبة ذات جودة يمكن معرفتها بقياس الطعم ، ورؤية اللون ، وساخنة أم لا ، وإذا كان فيها شيء غير مضبوط فإنه ليس هناك شيء يمكن عمله لضبط جودتها ، وعلى هذا فإنه يعاد التحكم في الكميات للمكونات والطريقة لضبط جودة الوجبة، ولإعداد وجبة ذات جودة يجب أن تمر عملية الإعداد بعدة مراحل بداية من ناحية إعداد المكونات وحتى الطهي بمراحله المختلفة ، وفي كل خطوة يمكن تصحيح الخطأ إذا ما تم شيء غير صحيح على سبيل المثال يمكن شراء مكونات جديدة إذا ما اتضح أن أحدها فاسد أو غير طازج أو أننا نعيد غسيل أحد المكونات إذا ما أتضح أن نظافته غير كافية ، وهكذا فإن جودة الوجبة يمكن تأكيدها بعمل اختبارات خلال عملية الإعداد، ويمكن تطبيق نفس المبدأ في عملية تطوير البرمجيات حيث يمكن إجراء قياسات ومراجعات مماثلة للتأكد من جودة البرمجية المنتجة. وكذلك فإنه لإعداد وجبه يلزم (وصفة للإعداد) والتي يمكن اتباعها وهذا مماثل تماما لاستخدام وسائل وطرق جيده خلال عملية تطوير البرمجية.

تدريب (1)



إذا ما طلب منك تطوير برمجية تقوم بعملية تحكم آلي لطـــائرة ما هي العوامل الهامة التي يجب أن تؤخذ في الاعتبار؟

تدريب (2)



ما هي معاملات الجودة التي تساعد قياس كثافة الخطأ (Fault Density) في تحقيقها؟

أسئلة تقويم ذاتي

عرف مصطلح الجودة وفقا لمقياس آيزو المعياري، وضح ذلـــك بأمثلة على:

- جودة التصميم.
- جودة التوافق.

حدد بإيجاز أهم السمات التي تحدد مدى جودة البرمجيات.

ما هي أهم أنشطة إدارة جودة البرمجيات.

2. إدارة جودة البرمجيات

(Software Quality Management)

إدارة جودة البرمجيات تشمل الأنشطة (Activities) أو العمليات (Processes) التي تضمن أن المشروع البرمجي يحقق أهدافه ، وبمعنى أخر فإن المشروع البرمجي يحقق توقعات العملاء، ويجب أن تنفصل إدارة الجودة عن إدارة المشروع لضمان الاستقلالية.

1.2 أنشطة إدارة جودة البرمجيات

(Software Quality Management Activities):

يمكن تلخيص أنشطة إدارة البرمجيات في التالي :

1.1.2 نشاط تخطيط الجودة

(Quality Planning Activity)

هي عملية وضع خطة تضمن جودة البرمجية المطلوب تطويرها وذلك من خلال الإجابة على الأسئلة التالية :

- كيف يتم تقدير الجودة المطلوبة للبرمجية تحت التطوير؟.
- ما هي السمات المؤثرة في جودة البرمجية؟.
- كيف يتم تقويم الجودة؟.
- ما هي المعايير القياسية التنظيمية التي يجب تطبيقها عند اختبار جودة البرمجية ؟ ، وما هي المعايير الجديدة التي يجب أخذها في الاعتبار عند الضرورة؟.

وتُعد مرحلة تخطيط الجودة أهم مراحل إدارة جودة البرمجيات حيث إن

التخطيط الجيد يضمن تحقيق المراحل التالية وتحقيق أهدافها ، ويجب أن تكون خطط الجودة قصيرة ومدونة في وثائق محددة موجزة حتى يمكن قراءتها جيداً. ويجب أن تشمل :

- غاية الوثيقة وأفقها ، وأهداف الجودة **(Quality Objectives)** .
- قوائم بجميع المعايير القياسية **(Standards)** الخاصة بالمشروع ، وبعملية البرمجة ، والمنتج نفسه ، وكيفية قياسها والتحقق من إنجازها.
- المسئوليات المحددة لأنشطة الجودة مثل : الفحوصات **(Inspections)** ، والمراجعات والتدقيقات **(Reviews and Audits)** ، والاختبارات **(Tests)** بما في ذلك اختبار التحقق والمصادقة ، والتوثيق **(Documentation)** ، والتحكم في العيوب وإجراء التصحيح في حالة حدوثها.
- الخطط المفصلة للأنشطة السابقة ، والتي يجب أن تنفذ متضمنة : الجداول الزمنية ، والموارد ، وتصديق المسئولين عليها.
- إدارة المخاطر **(Risks Management)**.
- الطرق والأدوات التي يجب أن يتم توظيفها لإتمام النشاطات والمهام المتعلقة بخطة تأمين جودة البرمجيات.

ويتمثل دخل مرحلة التخطيط في :

- سياسية جودة المنظمة **(Organization 's Quality Policy)**.
- المعايير القياسية للمنظمة **(Organization Standards)**.
- المعايير القياسية الصناعية وثيقة الصلة بالمشروع **(Relevant Industry Standards)**.
- التنظيمات **(Regulations)**.
- أفق عمل المشروع **(Scope of Project Work)**.
- متطلبات المشروع **(Project Requirements)**.

ويتمثل خرج مرحلة التخطيط في :

- المعايير القياسية المُعرفة للمشروع **(Standards Defined for the Project)**.
- خطة الجودة **(Quality Plan)**.

2.1.2 نشاط تأمين جودة البرمجيات

: (Software Quality Assurance Activity)

- **مبادئ وتعريفات :**

تُعرف عملية تأمين جودة البرمجيات بأنها : منهجية مخططة ونظامية **(Planned and Systematic Approach)** لتقويم الجودة **(Evaluation of the Quality)** ، وتحديد مدى الالتزام بالمقاييس المعيارية **(Standards)** في إنتاج المنتج البرمجي ، وكذلك العمليات **(Processes)** ، والإجراءات **(Procedures)** الخاصة بذلك ، وتشمل ، أيضا ، عملية التأكيد من أن المعايير والإجراءات قد تم تأسيسها والالتزام باتباعها خلال دورة حياة تطوير المنتج، وتتم عملية التأكد هذه من خلال أنشطة مراقبة العمليات **(Processes Monitoring)** ، وتقويم المنتج **(Product Evaluation)** ، ومراجعة وتتبع (تدقيق) **(Auditing)** جميع النشطات والإجراءات التي تم اتباعها في تطوير المنتج البرمجي لتحديد مدى كفاءته لتحقيق كامل أهدافه.

ويمكن القول أيضاً إن تأمين جودة البرمجيات تعني التأكد من أن البرمجية تم إنتاجها لتلبي أهداف الجودة المحددة في خطة الجودة ، حيث تختلف هذه الأهداف من مشروع لآخر ، حيث إنه لا بد أن تكون هذه الأهداف قد تم انتقاؤها من قائمة معاملات الجودة المتعارف عليها في مجال البرمجيات، والتي تم ذكرها سابقاً، ولتحقيق هذه الأهداف فإنه لا بد أن تتم عملية الاختبارات خلا ل تطوير البرمجية للتأكد من صحة تنفيذ العملية.

- **المقاييس المعيارية وإجراءات تطوير البرمجيات :**

إن وضع المقاييس المعيارية والإجراءات لتطوير البرمجيات عملية حرجة جداً ، حيث إنها تمثل إطار العمل **(Framework)** الذي يتم من خلاله عملية تطوير المنتج البرمجي ، أي أنها تؤسس طرق وصفية **(Prescribed Methods)** لتطوير البرمجيات، وتُعد عملية توثيق المقاييس المعيارية والإجراءات والأنشطة الخاصة بها من خلال تعريفات صريحة يمكن قياس مدى الالتزام عملية مهمة جداً في عملية تأمين جودة البرمجيات

- **المقاييس المعيارية (Standards) :**

هي معايير مُؤسسة **(Established Criteria)** يتم من خلالها مقارنة المنتجات البرمجية ، وتعد أساس إدارة الجودة الفعالة ، وقد تكون هذه المعايير دولية أو قومية أو مؤسسية أو معايير مشروع، وتعرّف معايير المنتج القياسية الخصائص التي يجب أن تكون عليها مكونات البرمجية مثل أسلوب البرمجة الشائع ، وتعرف الحد أو المدى أو التفاوت لبعض الكميات المقيسة التي يمكن الرضوخ أو التسامح فيها. ومن المعايير القياسية للمنتج البرمجي :

1. **المعايير القياسية للتوثيق (Documentation Standards) :** وهي تشمل معايير بنية هيكل المستند ، ومعايير تحديثه ، ومعايير تنسيق المحتويات ، الخ
2. **المعايير القياسية للتصميم (Design Standards) :** وتشمل الشكل ومحتويات تصميم المنتج ، وكذلك الطرق التي تتم عن طريقها ترجمة متطلبات البرمجيات إلى التصميم المناسب لها.
3. **المعايير القياسية للشيفرة (Code Standards) :** وتشمل اللغة التي سيتم بها تشفير البرنامج ، والقيود المفروضة عليها ، حيث يتم فيها بيان توضيحي للهياكل القانونية الصحيحة **(Legal Language Structures)** واصطلاحات الأسلوب **(Style Conventions)** المستخدم في عملية البرمجة ، وهياكل

البيانات **(Data Structures)** ، وواجهات الاستخدام **(Interfaces)** ، وكذلك تشمل التوثيق الداخلي للشيفرة **(Internal Code Documentation)**.

وتُعد **المعايير القياسية آيزو (ISO)** من أهم المعايير القياسية الدولية لإدارة الجودة القابلة للتطبيق على نطاق المنظمات من التصنيع إلى الخدمات الصناعية. ومن خلال المعايير القياسية أيزو يمكن تعريف نظام تأمين الجودة **(Quality Assurance System)** بأنه : البنية التنظيمية والمسؤوليات والإجراءات وعمليات البرمجة والموارد اللازمة لإنجاز إدارة الجودة.

ويُعد مقياس **ISO 9001** هو مقياس تأمين الجودة المطبق في هندسة البرمجيات ، حيث يتضمن هذا المقياس عشرين متطلباً يجب توفرها لكي يتحقق تأمين الجودة فعلياً ، حيث تتضمن هذه المتطلبات البنية التنظيمية ، ونظام الجودة ، وعمليات البرمجة ، وعمليات التفتيش والاختبارات ، وأعمال مراقبة المنتج ، وأعمال التصحيح والوقاية ، وعمليات التوثيق ، وعمليات التخزين والتغليف والحفظ والتسليم ، والتدريب ، وخلافه من العمليات.

ولكي تحصل شركة برمجيات ما على شهادة **ISO 9001** ، عليها أن تعتمد مجموعة من السياسات والإجراءات التي تشمل المتطلبات المطلوب تحقيقها في مقياس **ISO 9001** ، وعليها أن تثبت أنها تطبق هذه السياسات والإجراءات فعلياً. ولمزيد من المعلومات حول مقياس **ISO 9001** يمكن الرجوع لصفحات الويب المختلفة ذات الصلة بالموضوع والمنتشرة عبر الإنترنت.

- **الإجراءات (Procedures) :**

هي معايير مُؤسسة يتم من خلالها مقارنة عمليات التطوير **(Development)** والتحكم بالعمليات **(Control Processes)** ، ويتم صياغتها في خطوات واضحة يمكن أن تتبع لتنفيذ عملية ، وكل العمليات لابد أن يكون لها إجراءات موثقة، ومن الأمثلة على العمليات التي تحتاج إلى إجراءات الاختبارات والفحوصات الرسمية

.(Formal Inspections)

ولكي تكون عملية تأمين جودة البرمجية مؤكدة فإنه لابد من التخطيط لها مسبقا من خلال التخطيط الكلي للمشروع حيث يتبع مدير التخطيط للمشروع الخطوات التالية :

1. يقرر **(Decide)** : ما هي معاملات الجودة الأكثر أهمية لهذا المشروع على سبيل المثال (الاعتمادية ، قابلية الصيانة) وهي مشابهة تماما في إعداد وجبة فإن الاهتمام ينصب على النكهة والقيمة الغذائية كمتغيرات هامة.
2. **الانتقاء (Selection)** : اختيار المقاييس والطرق الموافقة للأهداف المنتقاة ، على سبيل المثال استخدام مقياس التعقيد لقياس الصيانة.
3. **التجميع (Assembles)** : تجميع ما سبق في البندين السابقين ليمثل خطة تأمين جودة للمشروع.

وعلى المؤسسات منتجة البرمجيات أن تقنع المستهلك باستخدام طرق مؤثرة في عملية الإنتاج وذكر ماهية هذه الطرق المؤثرة ، إضافة إلى عمل بيان عملي **(Demonstration)** لتوضيح هذه الطرق وتأثيرها ، وفي هذه الحالة يمكن القول إن تأمين جودة البرمجية لا تشمل فقط تنظيم عملية إنتاجية البرمجية وفقا لعوامل جودة المشروع ، ولكن مع بيان كامل موثق لها.

- **أنشطة تأمين جودة البرمجيات (Software Quality Assurance Activities) :**

يوجد العديد من الأنشطة التي يجب أن يقوم بها فريق تأمين جودة البرمجيات لتطوير منتج عالي الجودة ، حيث تهتم هذه النشاطات بالتخطيط لضمان الجـــودة ، والتقويم ، والمراقبة ، ومتابعة السجلات ، والتحليل وإصدار التقـــارير ، وإجمـــالاً يمكن ذكر أهم هذه النشاطات فيما يلي:

1. تقويم المنتج **(Product Evaluation)** : وهو عبارة عن أحد أنشطة تأمين جودة

البرمجيات حيث يتم فيه التأكد من أن المعايير القياسية والإجراءات قد تم اتباعها بصورة صحيحة وتتوافق مع المعايير القياسية التي تم تحديدها في وثيقة خطة الجودة ، وأن المنتج يعكس طبيعة المتطلبات التي قد تم وصفها في خطة المشروع.

2. مراقبة العمليات **(Processes Monitoring) :** عبارة عن أحد أنشطة تأمين جودة البرمجيات حيث يتم فيه التأكد من أن الخطوات المناسبة لتنفيذ العمليات (الإجراءات) قد تم إتباعها بصورة صحيحة وتتوافق مع الإجراءات التي تم تحديدها في وثيقة خطة الجودة.

3. التدقيق والمراجعات **(Auditing)** : وهي تُعد من الأنشطة المهمة لضمان جودة البرمجيات حيث تنظر بعمق داخل العمليات والمنتج واستخلاص المعايير القياسية والإجراءات التي تم اتباعها ومقارنتها بما هو مدون بخطة تأمين جودة البرمجيات. أي أن الغرض الأساسي لعملية التدقيق والمراجعات هي مراجعة المنتج البرمجي ، والتأكد من أن المقاييس المعيارية والإجراءات المعتمدة ونقاط تأمين الجودة قد تم اتباعها وأخذها في الاعتبار ، وإرسال تقرير دوري بالنتائج لمدير المشروع.

نشاط مراجعات الجودة

(Quality Reviews Activity)

هي آلية لتفحص كامل وبدقة بالغة كل نظام البرمجيات ومستندات التوثيق المرتبطة به ، حيث يقوم مجموعة من الأشخاص بمراجعة الشيفرة والتصميم والمواصفات وخطط الاختبار والمعايير القياسية ، بهدف اكتشاف خلل أو عدم تماسك ومتانة النظام ، ويتم تودين استنتاجات هذه المراجعات وإرسالها لمؤلفها أو للشخص المسئول عن تصحيح هذه الأخطاء (إن وجدت). ويجب أن يكون فريق المراجعة صغيراً ، ويتضمن أعضاء من فريق المشروع قد تكون لهم مساهمة فعالة ، فمثلاً في حالة مراجعة تصميم لفرعية معينة يجب أن يكون المهندسون الذين صمموا هذه الفرعية من البرنامج من ضمن أعضاء الفريق ،

فقد يساهمون برؤى مهمة للواجهات البينية لهذا الجزء وقد لا يتم اكتشافها إذا اعتبر الجزء كنظام قائم بذاته، ويجب أن تكون المراجعات مبنية على مراجعة مستنديه وليست مقتصرة على المواصفات أو التصاميم أو الشيفرة فقط ، حيث إنه يجب أن تتم مراجعة وثائق المشروع مثل نماذج العمليات ، وخطط الاختبار، وإجراءات إدارة التصميم ، ومعايير العمليات وكتيبات التشغيل.

وهناك العديد من أنماط المراجعة التي يمكن القيام بها في إطار هندسة البرمجيات، ومن أهم هذه الأنماط المراجعات التقنية ذات الصفة الرسمية والتي تتم بناءً على طلب رسمي يتناول تصميم البرمجية بحضور العملاء وكوادر تقنية وإدارية.

وتُعد المراجعات التقنية الرسمية **(Formal Technical Review "FTR")** أحد الأنشطة المتعلقة بتأمين جودة البرمجيات ، كما تعتبر آلية فعالة لتحسين جودة المنتج البرمجي ، وذلك من خلال تحقيق الأهداف التالية :

- كشف الأخطاء الوظيفية والمنطقية بالبرمجية.
- حصر التحسينات التي يجب إدخالها على المنتج لزيادة جودته.
- تثبيت أجزاء المنتج التي لا تحتاج أية تحسينات.
- التحقق من تلبية البرمجية للاحتياجات المطلوبة منها.
- التأكد من توافق البرمجية مع المعايير القياسية المُعرَّفة سلفاً.
- توفير التناسق والانسجام في عملية تطوير البرمجية.
- جعل المشاريع أكثر قابلية للإدارة.

وبصفة عامة يوجد عدة أنواع من المراجعات **(Types of Reviews)** منها :

- فحوصات التصميم أو البرنامج **(Design or Program Inspection)** لكشف الأخطاء التفصيلية في التصميم أو في الشيفرة ، وفحص ما إذا كان قد تم اتباع

المعايير القياسية ، وتتم المراجعات بدلالة قائمة فحص تتضمن الأخطاء الممكنة.

- مراجعات تقدم سير المشروع **(Project Progress Reviews)** لتوفير معلومات عن مدى تقدم سير المشروع للإدارة المعنية ، حيث يتم مراجعة العمليات والمنتج في نفس الوقت ، وتهتم بالتكلفة والخطط والجداول الزمنية.
- مراجعات الجودة **(Quality Reviews)** للقيام بتحاليل فنية لمكونات المنتج أو الوثائق لاكتشاف الأخطاء (الاختلافات) بين المواصفات وبين تصميم المكونات ، وقد تهتم بموضوعات الجودة مثل البعد عن المعايير القياسية أو سمات الجودة الأخرى.

ولكن ما هي نتائج المراجعات التقنية الرسمية؟ .

في نهاية المراجعات يجب على أعضاء فريق المراجعات أن يقروا إحدى القرارات التالية :

- قبول المنتج دون تعديلات إضافية.
- رفضه بسبب أخطاء مؤثرة (وهذا يستلزم مراجعة جديدة بعد إصلاح هذه الأخطاء).
- قبول المنتج مؤقتاً بشرط تصحيح الأخطاء القليلة التي اكتُشفت ، ولكن بدون أن يستلزم ذلك إعادة المراجعة.

2.2 أهمية إدارة جودة البرمجيات

(Software Quality Management Importance):

تتركز أهمية إدارة جودة البرمجيات في تزويد الإدارة بالبيانات الكافية لتكون على دراية بجودة منتجها ، وبهذه الطريقة يتسنى للإدارة الرؤية العميقة والثقة بأن منتجها عن مستوى طموحها ، وفي حالة وجود أي مشاكل في عملية تأمين جودة المنتج سيكون دور الإدارة مواجهة تلك المشاكل وتخصيص الموارد اللازمة لحلها

وصولاً إلى الجودة المرجوة ، هذا بالإضافة إلى أن إدارة جودة البرمجيات تقلل من تكلفة فشل النظام **(System Failure Costs)**.

ويمكن تقسيم تكلفة فشل النظام إلى تكلفة إخفاق داخلية وأخرى خارجية. فالتكلفة الداخلية هي التي تتكبدها الشركة أو المؤسسة المنتجة عند اكتشاف العيوب في المنتج قبل شحنه للعميل ، وتشمل تكلفة :

- إعادة التشغيل.
- أصلاح العيوب.
- تحليل أنماط الفشل.

أما تكلفة الفشل الخارجي فهي التكلفة المتعلقة باكتشاف العيوب بعدما أصبح المنتج بين يدي العميل ، ويمكن أن تتضمن تكلفة :

- معالجة الشكاوى.
- إصلاح العيوب.
- إعادة المنتج أو استبداله.
- الدعم التقني المستمر.
- أعمال الكفالة.

وبصفة عامة تتضمن تكلفة الجودة **(Cost of Quality)** جميع تكلفة النشاطات المتعلقة بالجودة ومتابعتها، ويمكن توزيع تكلفة الجودة على ثلاث نواحي هي:

1. **تكلفة الفشل :** تم مناقشتها أعلاه ويمكن حذفها في حالة عدم وجود عيوب في المنتج قبل شحنه للعميل وبعده.
2. **تكلفة منع الأعطال :** وتتضمن تكلفة مايلي :
   - تخطيط الجودة.
   - المراجعات التقنية الرسمية.
   - أدوات الاختبار.

- التدريب.

3. **تكلفة التقويم :** وتتضمن تكلفة النشاطات اللازمة للحصول على رؤية عميقة لحالة المنتج ، ومن أمثلة تكلفة التقويم مايلي:
   - الفحوصات المختلفة.
   - معايرة التجهيزات وصيانتها.
   - الاختبارات.

3. مقاييس البرمجيات

(Software Metrics)

إن الطريقة المنطقية الوحيدة لتحسين أي عملية برمجة هي قياس السمات المميزة لها ، وتطوير مجموعة من المقاييس **"Metrics"** ذات مغزى تستند إلى هذه السمات ، ومن ثم استخدام هذه المقاييس في توفير مؤشرات تقود إلى وضع استراتيجية للتحسين.

وبديهياً ، يمكن تخمين أن مقاييس البرمجيات **"software metrics"** متعلقة بالأعداد وقياس السمات المختلفة لعملية تطوير البرمجيات ، ولكن لإعطاء فكرة دقيقة عما تعنيه مقاييس البرمجيات فإننا نحتاج إلى تعريف واضح ، وإذا رجعنا إلى ما تم نشره بهذا الخصوص فسنجد فيه العديد من التعريفات والتي تحمل في معظم الأحيان نفس المضمون.

1.3 تعريف مقاييس البرمجيات

(Definition of Software Metrics)

لقد أخذ تعريف مقاييس البرمجيات أشكالاً متعددة منذ أن استهل ، نذكر منها :

- التعريف الذي اقترحه نورمان فونتن [6,7] حيث عرف مقاييس البرمجيات

بأنها تعبير متراكم **"Collective Term"** يستخدم لوصف المدى الواسع للأنشطة المتعلقة بالقياسات **"Measurements"** في هندسة البرمجيات. وهذه الأنشطة تتدرج من إنتاج الأعداد التي تصف سمات مصدر شيفرة البرمجية **"Software Source Code"** (وهي مقاييس البرمجة الكلاسيكية) إلى النماذج التي تساعد في التنبؤ بمتطلبات موارد البرمجية **"Software Resource Requirement"** وجودتها ، وكذلك السمات الكمية **"Quantitative Aspects"** الخاصة بتأمين الجودة ، وهذا يغطي أنشطة مثل تسجيل ومراقبة العيوب أثناء التطوير والاختبار.

- التعريف الذي اقترحه بول غودمان [6,8] حيث عرف مقاييس البرمجيات بأنها : تطبيق مستمر لتقنيات أساسها القياس **"Measurement-Based Techniques"** على عملية التطوير وما تنتجه من منتجات للحصول على معلومات إدارية ذات فائدة ومغزي بصفة دورية ، وذلك لتحسين العمليات ومنتجاتها.
- ومن منطلق أن الغرض الأساسي من استخدام مقاييس البرمجيات هو اتخاذ القرارات **"Decision Making"** والذي تم التأكيد عليه من قبل جرادي [9,10] حيث أشار إلى أن مقاييس البرمجيات تستخدم لقياس صفات معينة **"Specific Attributes"** لمنتج البرمجية أو عملية تطويرها للمساعدة في اتخاذ قرارات أفضل.
- أيضاً يمكن تعريف مقاييس البرمجيات بأنها [9]: الممارسة الفعلية لقياس الخصائص المختلفة لعملية تطوير البرمجية والمنتجات البرمجية للحصول على معلومات مفيدة وذات علاقة، لجعل العملية الإدارية تتم بكفاءة خلال كامل عملية التطوير.
- ووفقاً لمسرد تعريفات معهد المهندسين الكهربائيين والإلكترونيين"IEEE" الخاص بهندسة البرمجيات ، فإن مقاييس البرمجيات هي قياس كمي **"Quantitative Measure"** للدرجة التي يمتلك فيها نظام، أو مكون، أو عملية برمجة ما سمة

معينة.

من خلال ما سبق يمكن تعريف مقياس سمة معينة **"Specific Attribute Metric"** لنظام ، أو مكون ، أو عملية برمجة بأنه : مؤشر كمي **"Quantitative Indicator"** أو مجموعة من المؤشرات الكمية الناتجة عن تحليل مجموعة من البيانات المجمعة عن طريق الممارسة الفعلية لقياس هذه السمة على امتداد فترات طويلة من الزمن والربط بينها ، بحيث توفر هذه المؤشرات نظرة ثاقبة لمدير المشروع أو الممارسين ، أو مهندسي البرمجيات لاتخاذ قرار سليم لتحسين هذه السمة ، وإجراء المقارنات الموضوعية بين التقنيات والعمليات المستخدمة في عملية البرمجة، وبالتالي تحسين جودة المنتج أو العملية البرمجية على المدى القصير والبعيد.

فمثلاً يمكن أن نشتق مقياس لسمة مدى فعالية عملية البرمجية **" Programming Process Efficacy Attribute"** ، وذلك من خلال قياسات الأخطاء المكتشفة قبل إصدار البرمجية ، والعيوب المسلّمة للمستخدمين والتي يبلّغون عنها بعد إصدارها ، ومدى مطابقة البرنامج التنفيذي لتطلعات العميل ، والجهد الإنساني المستهلك ، والوقت المستهلك ، .... وغيرها من القياسات ، ومن ثم تحليل هذه القياسات والربط بينها للحصول على مقياس يتضمن مؤشرات كمية تُمكن المديرين والممارسين ومهندسي البرمجيات من اتخاذ القرار الصائب.

أسئلة تقويم ذاتي

لقد أخذ تعريف مقاييس البرمجيات أشكالا متعددة منذ ان استهل ما هو التعريف الذي يمثل خلاصة هذه التعريفات إجمالا من وجهة نظرك؟

2.3 لماذا مقاييس البرمجيات؟ (Why Software Metrics?)

تسمح لنا مقاييس البرمجيات بتفهم بيئة تطوير البرامج بشكل أفضل ، وتعطي معلومات مفصلة حول المشروع، وتستخدم المقاييس للتوقع ، وللإدارة ، ولتنظيم المنتج. ومع المقاييس الجيدة يمكن التخطيط للمشروع بطريق صحيحة، هذا وتبين مقاييس البرمجيات الخطوط العريضة المساعدة التي توضح أين نحن الآن وإلى أين نتجه خلال عملية التطوير، ويمكن تلخيص بعض النقاط التي توضح لماذا نحتاج مقاييس البرمجيات كما يلي [13] :

- بدون قياس عملية البرمجية أو جودة المنتج النهائي فإن التقدير الشخصي فقط محتمل ، وهو ليس مرغوباً فيه.
- بالمقاييس المتينة فإن المتطلبات يمكن أن يتم تعضيدها ، وبالتالي فإن أخطاء المكونات يمكن تشخيصها في المراحل المبكرة ، وبذلك يمكن تحسين أو ضمان تأمين الجودة.
- التنبؤ بمصدر المتطلبات فائدة أخرى مهمة لمقاييس البرمجيات.

وطريقة أخرى للإجابة عن السؤال هو تحديد المتاعب والمشاكل التي يمكن أن تنشأ في حالة عدم استخدام المقاييس في المشاريع البرمجية، وهنا نعرض ثلاث صعوبات تواجه المطورين والمدراء الذين لا يستخدمون مقاييس البرمجيات :

1. لا يمكنهم وضع أهداف يمكن قياسها للمنتج ، حيث إنهم لا يعرفون هل تم الوصول إليها أم لا ، وعلى سبيل المثال يمكن أن يَعدوا بأن المنتج سيكون سهل الاستخدام ، ويمكن الاعتماد عليه ، وسهل الصيانة، وحيث إنهم ليست لديهم تعريفات لهذه المقاييس فإن وصولهم لهذه الأهداف لا يمكن معرفته.
2. لا يستطيع المديرون قياس مكونات التكلفة ، وبالتالي فإنه من شبه المستحيل تقدير التكلفة الكلية ، وبذلك لا يمكن إعطاء تقدير دقيق للعميل.

3. المطورون والمديرون يخفقون في تحديد أو توقع جودة المنتج الذي يتم إنتاجه ، وبالتالي إذا احتاج العميل إلى معرفة مدى موثوقية المنتج أو كم العمل اللازم لتغيير المنتج ، فإنهم لا يستطيعون تزويده بإجابة.

ويمكن تلخيص أهمية مقاييس البرمجيات في أنها :

1. تساعد فيما يجب عمله خلال عملية التصميم ، وترشدنا إلى اختيار برمجية واضحة ، وبسيطة ، وسهلة التعامل ، وموثوق بها.
2. تقودنا إلى طريقة جيدة لقياس جودة البرمجة.
3. تساعدنا في التنبؤ بالجهد اللازم للتطوير.
4. تساعدنا في المفاضلة بين الطرق المختلفة لتطوير البرمجية.
5. تساعدنا في تحسين الممارسات العملية.

أسئلة تقويم ذاتي

وضح اهم النقاط التي توضح لماذا نحتاج لمقاييس البرمجيات.

3.3 خصائص مقاييس البرمجيات الجيدة

(Characteristics of Good Software Metrics) :

المقاييس الجيدة لا بد أن تسهل عملية تطوير النموذج القادر على توقع العملية أو متغيرات المنتج ، ليس فقط بالوصف ولكن يجب أن تشمل النقاط التالية :

- أن تكون سهلة ، مُعرفة بالضبط ، وبالتالي يكون من الواضح كيفية تقويم المقاييس.
- تكون موضوعية **"Objective"** إلى أقصى حد ممكن.
- يمكن الحصول عليها بسهولة وبتكلفة معقولة.
- قابلية للتطبيق بفاعلية ، يجب أن تقيس المقاييس الهدف المطلوب قياسه.

- متينة **"Robust"** ، لا تكون حساسة نسبياً أمام التغييرات البسيطة للعملية أو المنتج.
- يجب أن تشمل على قيم للبيانات الملائمة للكميات المقيسة.

أسئلة تقويم ذاتي

ماهي أهم خصائص مقاييس البرمجيات الجيدة؟.

4.3 تصنيف مقاييس البرمجيات [14 , 9]

(Classification of Software Metrics)

ووفقاً لما أورده إيفرالد [14] فإن مقاييس البرمجيات يمكن تقسيمها إلى : مقاييس للمنتج **"Product Metrics"** ومقاييس للعملية **"Process Metrics"**.

- مقاييس المنتج "Product Metrics" :

مقاييس المنتج تقيس البرمجية كمنتج ابتداءً من مراحله الأولية وحتى تسليمه للعميل ، وهي تتعلق كذلك بسمات : شيفرة المصدر **"Source Code"** ، وقياس المتطلبات **"Measure the Requirements"** ، وحجم البرنامج **"Size of Program"**، والتصميم **"Design"** وخلافه، وإذا ما تم تعريف مقاييس المنتج في المراحل الأولى من عملية التطوير فإن هذا سيكون مساعداً إلى حد كبير في التحكم في عملية تطوير المنتج. وسوف نتناول بعض من مقاييس المنتج ، مثل : مقاييس الحجم**"Size Metrics"** ، ومقاييس التعقيد **"Complexity Metrics"** ، ومقاييس هالستيد للمنتج **"Halstead's Product Metrics"** ، ومقياس الجودة **"Quality Metrics"** ، وذلك على سبيل المثال.

◆ مقاييس الحجم **"Size Metrics"** :

تُعد مقياس حجم البرمجية من أبسط المقاييس حيث يمكن حسابها كميا بطريقتين :
الأولى : (الحجم بعدد البايت) : ويرتبط هذا بذاكرة المعالج وحجم القرص الصلب مما يؤثر تلقائيا على أداء الحاسب.
الثانية : الحجم بعدد الجمل ، وذلك عن طريق قياس حجم البرمجية عن طريق قياس عدد سطور الشيفرة **"LOC"** ، أو حساب عدد نقاط الوظيفة **"FP"** " ، أو باستخدام طرق أخرى كما أوضحنا ذلك في الوحدة الخامسة. ويتم تحديد حجم البرمجية في المرحلة المبكرة للمشروع مما يكون له فائدة كبيرة ، حيث يرتبط هذا بجهد التطوير وتكاليف الصيانة.

◆ مقاييس التعقيد **"Complexity Metrics "** :

في الأيام الأولى للبرمجة كانت الذاكرة الرئيسية للحاسبات صغيرة وكانت المعالجات بطيئة للغاية وكان الهدف هو إنتاج برمجيات تعمل بكفاءة مما جعل فئة المبرمجين تلجأ للحيل **"Tricks"** للتغلب على هذه الصعاب مما زاد تعقيد إنتاج هذه البرامج ، أما في هذه الأيام فقد تغير الوضع حيث أصبح التركيز على خفض زمن تطوير البرمجية وسهولة صيانتها وأصبح التركيز كذلك على كتابة برامج سهلة ، ومفهومة ، وواضحة ، وبسيطة ، مع إمكانية تعديلها. وبالتالي يصبح من السهل اختبارها ، وفهمها وكذلك تعديلها.

ولكن ما هي أهم السمات الأساسية للبرامج البسيطة؟ :

- سهولة اكتشاف الأخطاء فيها.
- أسرع في الاختبار.
- يمكن الاعتماد عليها.
- أسرع في إجراء التعديلات عليها.

وبرغم المميزات سابقة الذكر للبرمجيات البسيطة إلا أن كثيرا من مصممي البرمجيات غالبا ما يسلكون سلوكاً معاكساً تماما حيث يبتعدون تماما عن الحلول البسيطة بغرض الكسب من تعقيدات برامجهم، إلا أن هذا لا يعني أن هناك كثيرا من محللي ومصممي البرمجيات في هذه الأيام يبذلون قصارى جهدهم لتطوير برمجيات تتميز بكل من الوضوح والبساطة. وقد ينهي المبرمج تطوير البرمجية معتقدا أنها تعمل بصحة ومكتوبة بوضوح ولكن كيف نعرف أنها مكتوبة بوضوح؟ هل البرنامج القصير يلزم بأن يكون أبسط من البرنامج الطويل لنفس الهدف؟ هل البرمجية المتداخلة في كتابتها أبسط من البرمجية المعدة بدون تداخل؟ هذه أسئلة يهتم بها الناس وحتى الآن ليس لها إجابات محددة واضحة.

وللوصول إلى تعريف مقياس مفيد لقياس سمة التعقيد دعنا نضع مجموعة من الفروض التي تؤثر في تعقيد برنامج معين فمنها على سبيل المثال طول البرنامج ، وعدد المسارات خلال البرنامج ، وأماكن استقبال البيانات ، حيث يمكن استخدام هذه العوامل في الحصول على معادلة لتعريف سمة التعقيد بحيث يمكن اختبار هذه المعادلة ودراسة مدى خرج هذه المعادلة والوقت المأخوذ لكتابة أو فهم برنامج ، حيث إنه باستخدام مقاييس التعقيد يمكن إدارة وتنظيم عملية تطوير البرمجية وقياس مدى التعقيد فيها ، ومن أشهر هذه المقاييس :

- مقياس تعقيد الانحناءات **"Cyclomatic Complexity Metric"** : لأي برمجية يمكن أن ترسم له خريطة تدفق التحكم البيانية **"Control Flow Graph"** (G) ، حيث تمثل كل عقدة **(Node)** فيه كتلة من الشيفرة المتسلسل **"Block of Sequential Code"** ، وكل قوس يمثل تفرع أو نقطة قرار **"Decision Point"** في البرنامج ، وباستخدام نظرية الرسم يمكن اكتشاف مدى تعقيد البرنامج ، والذي يقيس مباشرة عدد المسارات الخطية غير المعتمدة **"Linearly Independent Paths"** بشيفرة البرنامج **"Program's Source Code"** ، وذلك وفقاً للعلاقة

الرياضية التالية :

$$V(G) = E - N + 2$$

حيث **V(G)** تمثل مدى تعقيد خريطة تدفق تحكم البرنامج **(G)** ، **"E"** تمثل عدد المسارات القوسية **"Arcs"** (أسهم التوصيل) بخريطة التدفق ، و **"N"** عدد العقد بالرسم البياني.

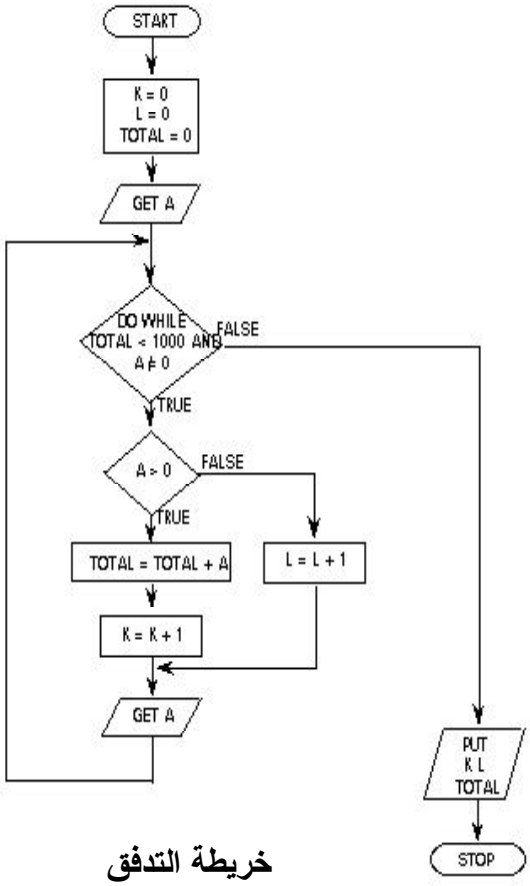
وللبرامج الهيكلية **"Structured Program"** يمكن حساب مدى تعقيد خريطة تدفق تحكم البرنامج V(G) مباشرة بحساب عدد نقاط القرارات في جسم البرنامج ، وذلك طبقاً لاقتراح ماكابي لسمة التعقيد **"McCabes Cyclomatic Complexity"** [15] ، حيث اقترح بأن سمة التعقيد لا تعتمد على عدد الجمل ولكن تعتمد اعتماد مطلقاً على هيكلية جمل القرارات **(Decision Statements Structure)** في بنية البرمجية (مثل عدد جمل : **If** و **While** ، والجمل المشابهة). ولحساب التعقيد وفقا لهذه الفرضية يكون رقم التعقيد = عدد الحلقات المغلقة + 1

(Cyclomatic Number) = "No. of Closed Loops" + 1
Or = " No. of Decision Points " + 1

ومقياس ماكابي لتعقيد الانحناءات **"McCabe's Cyclomatic Complexity Metric"** يُعد مؤشراً للجهد المبذول في عملية البرمجة **"Programming Effort"** ، **وللجهد المطلوب لعملية الصيانة "Maintenance Effort"** ، ولأداء عملية فحص وتحديد موضع الأخطاء وتصحيحها في شيفرة البرنامج (تنقيح البرنامج) **" Debugging Performance ".**

تدريب (3)

موضح أدناه خريطة التدفق لبرنامج ما ارسم خريطة تدفق التحكم الخاصة به ، ومن ثم أوجد درجة تعقيد البرنامج طبقاً لمقياس ماكابي (McCabe).



خريطة التدفق

Flowchart

وهناك طريقتان لاستخدام مقياس ما كابي لدعم تطوير البرمجيات :

الأول : إذا كان لدينا خوارزميتان **"Two Algorithm"** لحل نفس المشكلة فإنه يمكن استخدام هذا المقياس لاختيار الخوارزمية الأبسط والأبسط.

الثاني : على حسب فرضية ماكابي إذا كان مقياس التعقيد لجزئية يزيد عن 10 فإنه يعتبر كبير جدا ، وفي هذه الحالة يقترح ما كابي إعادة كتابة البرمجية مرة أخرى

أو يمكن تقسيمها إلى عدة أجزاء صغيرة.

إلا انه يعاب على المقياس السابق النقاط التالية :

1. لماذا تم اختيار الرقم 10 كحد أقصى أو كحد نهائي لقياس التعقيد؟ ما هي الأسس التي تم عليها اختيار هذا الرقم.
2. لم يضع المعيار حد محدد لطول البرمجية ، حيث إنه وفقاً لهذا المقياس فإن تعقيد برمجية طولها صفحة واحدة بدون قرارات يساوي تعقيد برمجية طولها العديد من الصفحات المماثلة.
3. يعتمد المقياس كلية على تنظيم السريان للجمل (جمل التوجيه) مهملا البيانات وإدخالها ، فهناك برنامج يعتمد على مداخل قليلة للبيانات مباشرة وآخر عكسه تماما يحوي مداخل كثيرة للبيانات من مداخل غير مباشرة.

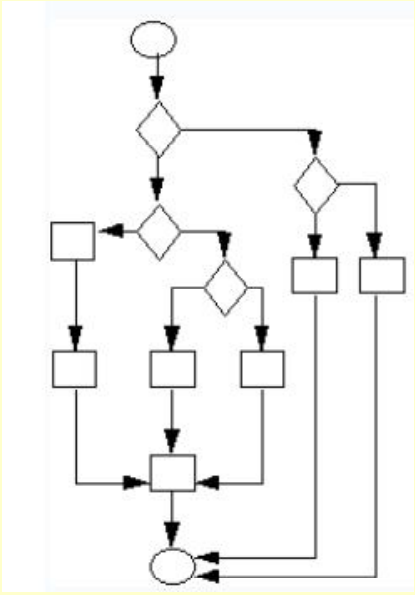
ومن هنا يتضح إهمال قياس ماكابي لعوامل أخرى يمكن أن تؤخذ في الاعتبار بغرض تحسينه ، إلا أنه وحتى الآن يُعد مقياساً مهماً جدا وله تأثير فعال ويمكن الانطلاق منه لتطوير معادلات أكثر دقة أخذه في الاعتبار العوامل الأخرى التي تم الإشارة لها.

ويمكننا القول إن وجود مقياس لسمة التعقيد يساعد مطوري البرمجيات في الحيثيات التالية :

- تقدير أو تخمين الجهد اللازم لعمل جزئية.
- اختيار أبسط التصميمات من التصميمات المتاحة والمرشحة.
- إرسال إشارة حينما تكون هناك جزئية مركبة والتفكير في إمكانية تجزئتها أو تقسيمها.

تدريب (4)

الشكل التالي يوضح خريطة تدفق التحكم لبرنامج "P" ، أحسب درجة تعقيد هذا البرنامج.



خريطة تدفق التحكم للبرنامج "P"

تدريب (5)

للإجرائية التالية ، ارسم خريطة تدفق التحكم ، ومن ثم احسب درجة تعقيدها.

```
Procedure sort (var x: array of integer; n: integer);
var i, j, save : integer;
begin
(1.)    for i:=2 to n do                    (node 1)
(2.)    for j:=1 to i do                    (node 2)
(3.)        If x[i]<x[j] then               (node 3)
(4.)        Begin                           (node 4, including rows 4-7)
(5.)            save:=x[i];
(6.)            x[i]:=x[j];
(7.)            x[j]:=save;
(8.)        end                             (node 5)
(9.)  end;                                  (node 6)
```

◆ مقاييس هالستيد للمنتج **"Halstead's Product Metrics" [14, 15] :**

اقترح هالستيد مجموعة من المقاييس والتي يمكن تطبيقها على العديد من المنتجات البرمجية حيث ناقش مفردات البرنامج (n) **"Program Vocabulary"** ، وطول البرنامج (N) **"Program Length"**، وكذلك حجم البرنامج (V) **"Program Volume"** ، وذلك طبقاً للعلاقات الرياضية التالية :

- مفردات البرنامج = $n = n1 + n2$
- طول البرنامج = $N = N1 + N2$
- حجم البرنامج = $V = N \log_2 n$ ، وهو يمثل قياس حجم الذاكرة المطلوب (مقيساً بالبايت) لتخزين البرنامج .

حيث إن :

n1 : عدد العوامل المفردة **(Unique Operators)** في البرنامج ، حيث يمكـــن أن تكون هذه العوامل حسابية أو منطقية ، تلك التي تستخدم في عملية البرمجة ، مثـــل الموضحة بالجدول رقم 8.1.

جدول رقم 8.1 : عوامل هالستيد (Halstead' Operators)

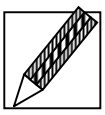
وصف العوامل	مثال
العوامل الحسابية والمنطقية **Mathematical and Logical Operators**	*** , + , - , / , ( ) , { } , = , > , <** **, ……..**
الكلمات المحجوزة **Reserved Words**	**If , goto , case, while ,** **…….**
محددات النوع **Type Qualifiers**	**Const , friend , volatile ,** **Double , ……**
مخصصات طبقة الذاكرة **Storage Class Specifiers**	**Static , Typedef, ……**

n2 : عدد المعاملات المفردة **(Unique Operands)** في البرنامج ، حيث يُعــرف المعامل **(Operand)** بأنها هدف العملية الحسابية أو المنطقية أو التعليمية البرمجيــة ، فعلى سبيل المثل : في الجملة X = 2 + 3 : X ، 2 و 3 تُعــد معــاملات ، = و + هما عاملان.

N1 : عدد العوامل الكلية **(Total Number of Operators)** في البرنامج .

N2 : عدد المعاملات الكلية **(Total Number of Operands)** في البرنامج .

تدريب (6)



أوجد حجم الذاكرة المطلوب لتخزينه لتخزين المقطع البرمجي التالي

```
if (k < 2)
   {
      if (k > 3)
      x = x*k;
   }
```

◆ مقاييس الجودة **"Quality Metrics" :**

تعتمد جودة البرمجيات على عدة سمات منها : الصحة ، والكفاءة ، وإمكانية الحمل ، وإمكانية الصيانة ، والاعتمادية ، والمتانة ، وغالباً ما يتم عمل مقاييس الجودة خلال كل مرحلة من دورة التطوير ، ومن هذه المقاييس :

1. مقاييس الكشف **"Detect Metric"** : عدد الأخطاء في المنتج البرمجي يجب أن يكون قابلاً للاشتقاق بسهولة ، ويمكن اعتباره مقياساً للمنتج ، وقد اقترح إيفرلاد [14] بعض الإجراءات التي تتبع لحساب العيب في المنتج منها :

- عدد تغييرات التصميم.
- عدد الأخطاء التي تم كشفها أثناء فحص الكود.
- عدد الأخطاء التي تم كشفها أثناء اختبارات البرنامج.
- عدد تغييرات الكود المطلوب.

وباستخدام قيم هذه المقاييس يمكن التنبؤ بجودة المنتج.

2. مقاييس الموثوقية (الاعتمادية) **"Reliability Metrics"** : تمثل موثوقية البرمجيات أحد العناصر الأساسية في جودته الإجمالية ، إذ لا يهمنا كثيراً أن تستوفى البرمجية بعض شروط الجودة المختلفة إن كانت تُخفق في أداء المهمة المطلوبة منها إخفاقاً متكرراً. وموثوقية البرمجية قابلة للقياس والضبط ، وتقديرها ممكن باستخدام البيانات التاريخية ، وتلك التي تتراكم أثناء عملية التطوير. ويمكن تعريف موثوقية برمجية ما بأنها [1] : "احتمال أن يعمل البرنامج بدون عطل في بيئة محددة وخلال زمن معين". لتوضيح ذلك لنفرض أن موثوقية البرنامج **"P"** تُقدر بـــ 96% لعمل مدة 8 ساعات ، فهذا يعني أنه إذا قمنا بتشغيل البرنامج **"P"** مئة مرة ، وكانت مدة المعالجة المطلوبة تمتد إلى ثماني ساعات ، فإن من المتوقع أن يجري التنفيذ بدون أعطال في 96 مرة من المرات المئة.

ويُعد معامل قياس كثافة الأخطاء (Fault Density "FD" Parameter) من مقاييس الموثوقية التي تعتمد على احتساب نسبة العيوب في ألف سطر شيفرة **(Defects/KLOC)** ، حيث تكتشف الأخطاء خلال التحقق والاستخدام العادي للبرمجية، وهناك بعض الأخطاء يمكن تصحيحها ولهذا لا تدخل في حساب المعامل **"FD"**. وهناك أخطاء أخرى في البرمجيات لا تكتشف ويُعد الحد (2 – 50) مقياساً مناسباً لكل KLOC والحد 2 بالضبط الحد الأمثل. وهذا المقياس مفيد جدا إلى حد كبير في زيادة تأثير طرق التحقيق وقياس الصحة **(Correctness)** في تأمين جودة البرمجيات، ولقد أثبتت معظم الدراسات العملية أن معظم الأخطاء نادرا ما تسبب فشل للنظام في حين أن عدداً صغيراً من الأخطاء قد يسبب فشل النظام ، ولهذا يكون من الأجدى بذل جهد لاكتشاف هذا العدد الصغير من الأخطاء.

إحدى التحديات الرئيسية في استخدام مقياس كثافة الخطأ هو أن بعض الأخطاء تكون واضحة خلاف الأخطاء الأخرى ، ولهذا فإن المقياس الأكثر فائدة للمستخدمين هو **مقياس المتوسط الزمني بين الأعطال (Mean Time Between Failures (MTBF))** ، **حيث :**

$$MTBF = MTTF + MTTR \ldots\ldots\ldots\ldots\ldots (1)$$

"MTTF" : مقياس المتوسط الزمني للعطل (Mean Time to Failure (MTTF)) ، وهو يعبر عن المتوسط الزمني للنظام للأداء بدون فشل، وهذا يمكن قياسه بتسجيل الأوقات التي تحدث فيها الأعطال وحساب متوسط الوقت بين الأخطاء المتتابعة. وهذا المقياس يمكن أن يعطي تنبؤاً لمعدل الخطأ المستقبلي للنظام.

"MTTR" : مقياس المتوسط الزمني للتصليح (Mean Time to Repair (MTTF)) ، وهو يعبر عن المتوسط الزمني لإصلاح الأعطال.

إضافة إلى وجود مقياس للموثوقية ، لا بد من تطوير مقياس للتوفر **(Availability)** ، والذي يقيس مدى احتمالية أن تعمل البرمجية في وقت معين بشكل مطابق للمتطلبات الموضوعة وبدون أعطال. ويتم احتساب قيمة مقياس التوفر **"AV"** طبقاً للمعادلة التالية :

$$AV = [MTTF / (MTTF + MTTR)] * 100\% \ldots\ldots\ldots (2)$$

نلاحظ من المعادلة (1) أن المقياس **MTBF** ذو حساسية متساوية لكل من **MTTF** ، و **MTTR** ، بينما نلاحظ من المعادلة رقم (2) أن المقياس **AV** ذو حساسية أكبر تجاه المقياس **MTTR** والذي يمكن اعتباره قياساً غير مباشر لقابلية صيانة البرمجية.

◆ **مقاييس العملية (Process Metrics) :**

تقيس هذه المقاييس خواص إجراء عملية تطوير البرامج : مثل الوقت ، وعدد الأشخاص المشتركين في التطوير ، وأنواع المنهجيات. وهنا ينصب التركيز على السمات المناسبة لبناء النموذج الذي منه توقع وتنظيم خطة التطوير، وتستخدم هذه المقاييس لتحسين العملية وإدارة المتطلبات والتي يتم تنفيذها في عملية التطوير لإنتاج منتج ذي جودة عالية ويحقق رغبات العميل.

وقد قسم جرادي [11] مقاييس البرمجيات من حيث طريقة القياس إلى نوعين

◆ **المقاييس البدائية "Primitive Metrics" :**

وهي مقاييس يمكن قياسها مباشرة **"Directly Measure"** ، مثل عدد سطور البرمجية **"LOC"** ، أو عدد الأخطاء في البرنامج.

◆ **المقاييس المحسوبة "Computed Metrics" :**

وهي مقاييس يمكن قياسها بطريقة غير مباشرة مثل جودة المنتج **"Quality of Product"** عن طريق قياس مقاييس أخرى .

وطبقاً لما جاء في [3] يمكن أن تقسم مقاييس البرمجيات إلى مقاييس ديناميكية ، ومقاييس استاتيكية.

◆ المقاييس الديناميكية "Dynamic Metrics" :

وهي مقاييس يتم تجميعها بواسطة قياسات يتم إجراؤها على البرمجية أثناء التشغيل **(Execution)** ، لذا فهي تتعلق كثيراً بسمات جودة البرمجيات **(Software Quality Attributes)** ، حيث يتم قياس زمن استجابة النظام **(Response Time of a System)** ، وسمة الأداء **(Performance Attribute)** ، وعدد الإخفاقات **(Number of Failures)** ، وسمة الاعتمادية **(Reliability Attribute)** .

◆ المقاييس الاستاتيكية "Static Metrics" :

وهي مقاييس يتم تجميعها بواسطة قياسات يتم إجراؤها على البرمجية أثناء وضع عدم التشغيل ، ولذا فهي تتعلق بصورة مباشرة بتقويم سمات قابلية التعقيد وقابلية الفهم والصيانة.

هذا الجدول التالي يلخص أشهر المقاييس المستخدمة في عملية اختبار البرمجيات:

المقياس	الوصف
دليل رضاء العميل Customer Satisfaction Index	**هذا الدليل يتم طرحه قبل وأثناءه وبعده تسليم المنتج وهو عبارة عن استمارة استبيان تشمل العديد من الأسئلة بغرض تحليل :** • **عدد طلبات تحسين النظام في السنة.** • **عدد طلبات الصيانة للنظام في السنة.** • **عدد طلبات المساعدة عن طريق الخدمة الساخنة.** • **وقت التدريب لكل مستخدم جديد.** • **عدد مرات الطرح للنظام من قبل المنتج.**

المقياس	الوصف
كمية الأعطال المستلمة Delivered Defect Quantities	وهي تحسب لكل نقطة دالة أو لكل عدد سطور من الشيفرة في وقت التسليم (الأشهر الثلاثة الأولى أو السنة الأولى من التشغيل مع تحديد مستويات الخطورة حسب التصنيف أو السبب(عطل متطلبات– عطل تصميم – عطل مساعدة أو توثيق).
مدى الاستجابة Responsiveness	يعبر عنه بالوقت المأخوذ لتصحيح العطل حسب أولويته ومستوى خطورته.
عدم ثبات المنتج. Product Volatility	ويعبر عنه بالنسبة بين عدد طلبات الصيانة لجعل النظام يلبي المواصفات إلى طلبات تحسين أداء النظام
نسب الأعطال. Defect Ratios	العيوب التي توجد بالمنتج بعد تسليمه لكل نقطة داله. العيوب التي توجد بالمنتج بعد تسليمه لكل عدد سطور من الشيفرة. عيوب قبل التسليم: عيوب ما بعد التسليم السنوية.
كفاءة إزالة العيوب. Defect Removal Efficiency	• عدد العيوب ما بعد التسليم والتي تكتشف من قبل الزبائن العاملين في الحقل. • عدد العيوب التي تكتشف داخليا بالتفتيش قبل التسليم بالنسبة للعدد الكلي للأخطاء • العدد الكلي للعيوب وهو عبارة عن مجموع العيوب السابقة (الداخلية والخارجية) في السنة الأولى بعد تسليم المنتج.
تعقيد المنتج المسلم Complexity of Delivered Product)	• يستخدم مقياس ماكابي. • توقع العيوب وتكاليف الصيانة استنادا على إجراءات التعقيد.
تغطية الاختبار. Test Coverage	• النسبة المئوية للمسارات ، التفريعات ، الحالات التي تم اختبارها. • النسبة بين عدد العيوب المكتشفة إلى عدد العيوب المتوقعة.
تكلفة العيوب. Cost of Defects	• فاقد العمل لكل عطل يحدث أثناء التشغيل. • تكلفة توقف العمل • فاقد المبيعات والنية الحسنة.

المقياس	الوصف
	• تكاليف المقاضاة الناتجة من العيوب • تكلفة الصيانة السنوية لكل نقطة داله. • تكلفة التشغيل السنوية لكل نقطة داله.
تكلفة جودة النشاطات. Costs of Quality Activities	• تكاليف المراجعات والفحوصات والإجراءات الوقائية. • تكاليف التخطيط للاختبار والإعداد له. • تكاليف إجراء الاختبار وتتبع العطل. • تكاليف تشخيص العيوب وإصلاحها. • تكاليف الوسائل المساعدة. • تكاليف الاختبار والأسئلة المساعدة المصاحبة للمنتج.
إعادة العمل. Re-work	• جهد إعادة العمل بالساعات كنسبة من ساعات التشفير الأصلية. • عدد السطور من الكود المعادة بالنسبة لعدد السطور الكلية المسلمة. • عدد المكونات المعادة بالنسبة لعدد المكونات المسلمة.
الإعتمادية ومقاييس تقويم اختبارات تطبيق النظام. Reliability	تعرف على أنها النسبة بين الوقت الذي فيه النظام متاح إلى الوقت الذي يحتاجه النظام لكي يكون متاحاً. وتقاس بالمقاييس التالية : • متوسط الوقت بين الفشل • متوسط الوقت للتصليح • عدد مرات إعادة طلب النظام أو الإصدارات المعدلة منه.

أسئلة تقويم ذاتي

إرسم جدولاً لخص فيه أشهر المقاييس المستخدمة في عملية اختبار البرمجيات.

5.3 مقياس تحديد مستوى مؤسسات تطوير البرمجيات

يتم تحديد مستوى مؤسسات تطوير البرمجيات عن طريق مقياس يحتوي على خمسة مستويات ، ويعتمد على ما يُسمى بنموذج نضج القدرة **"Capability Maturity Model (CMM)"** ، وذلك بطرح استمارة استقصاء تم تطويرها خصيصا لهذا الغرض من معهد هندسة البرمجيات **(Software Engineering Institute "SEI")** من جامعة ميلون بالولايات المتحدة **(Carnegie Mellon University "CMU")** ، ويمكن تعريف هذه المستويات كالتالي :

- **المستوى الأول (بدائي) (Initial)** : وفيه يتم إنتاج البرمجية بطريقة عشوائية بعمليات بسيطة غير منظمة حيث يعتمد نجاحها على جهد أشخاص محددين وذوي مهارات داخل المؤسسة وليس على طريقة إدارة المؤسسة للعمل.

- **المستوى الثاني (مثبت الجدارة) (Repeatable)** : وفيه تتم إدارة المؤسسة لإنتاج برمجية تراعي فيها كل من التكاليف ، الجدول الزمني والأداء وفيه يمكن أن تثبت المؤسسة نجاحها لنفس نوعية المشاريع التي تقوم بتنفيذها.

- **المستوى الثالث (معرف) (Defined)** : وفيه تتم عملية التطوير لكل من الإدارة وأنشطة هندسة البرمجيات معرفة وموثقة ويراعى فيها القياسية في عملية الإنتاج ، إضافة إلى ما تم توضيحه في المستوى الثاني.

- **المستوى الرابع (مهيأ إداريا) (Managed)** : وفيه يراعى تفاصيل القياسات لعملية التطوير للمنتج مجمعة ومحددة كميا بطريقة منظمة إداريا ، ويشمل كذلك ما تم توضيحه في المستوى الثالث.

- المستوى الخامس (الأمثل) **(Optimizing)** : وفيه تتم عمليات القياس باستمرار لتحسين أداء المنتج حيث يتم اقتراح طرق أخرى ووسائل للاختبار بغرض التحسين وكذلك ما تم طرحه في المستوى الرابع.

ويحبذ في كل الأحوال شراء البرمجيات من المؤسسات ذات المستوى الخامس والتي غالبا ما تصنع برمجيات ذات جودة عالية بأقل تكاليف.

أسئلة تقويم ذاتي

يشمل مقياس تحديد مستوى مؤسسات تطوير البرمجيات خمسة مستويات، ماهي هذه المستويات، ولماذا يحبذ في كل الأحوال شراء البرمجيات من المؤسسات ذات المستوى الخامس.

الخلاصة

اشتملت هذه الوحدة على :

– السمات العامة لجودة البرمجية هي أنها:

تلبي احتياجات المستخدم ، وهي خالية من العيوب ، يمكن الاعتماد عليها في العمل كما يمكن إجراء الصيانة لها.

– تعريف الجودة "وفقاً لمقياس آيزو المعياري 8204 " هي، المجموع الكلي لميزات وخصائص منتج أو خدمة ما والتي تؤثر على قدرته/قدرنها على تلبية حاجات المستخدم المحدودة أو الضمنية.

– إدارة جودة البرمجيات : تشمل الأنشطة أو العمليات التي تضمن أن المشروع البرمجي يحقق أهدافه وهذه الأنشطة هي نشاط تخطيط الجودة ونشاط تأمين جودة البرمجيات ، نشاط مراجعات الجودة.

– أهمية إدارة جودة البرمجيات

تقسيم تكلفة فشل النظام إلي تكلفة إخفاق داخلية وتشمل إعادة التشغيل ، وإصلاح العيوب ، وتحليل أنماط الفشل .

تكلفة فشل خارجي وتتضمن معالجة الشكاوي ، إصلاح العيوب ، إعادة المنتج أو استبداله ، الدعم التقني المستمر. وأعمال والكفالة توزيع تكلفة الجودة على ثلاث نواحي هي وتكلفة الفشل وتكلفة منع الأعطال ، وتكلفة التقويم.

– مقاييس البرمجيات : تنقسم إلى مقاييس المنتج مثل مقاييس الحجم ومقاييس التعقيد ، ومقاييس هالستيد للمنتج ، ومقاييس الجودة.

مقاييس العملية : تقيس هذه المقاييس خواص إجراء عملية تطوير البرامج مثل الوقت وعدد الأشخاص المشتركين في التطوير وأنواع المنهجيات.

مقاييس تحديد مستوى مؤسسات تطوير البرمجيات وهذه المستويات هي :

المستوى الأول (بدائي) وفيه يتم إنتاج البرمجية بطريقة عشوائية ، المستوى الثاني (مثبت الجدارة) وهنا تتم إدارة المؤسسة لإنتاج برمجية يراعى فيها التكاليف، والجدول الزمني والإفراد والمستوى الثالث (معرف) وفيه تتم عملية التطوير لكل من الإدارة وأنشطة هندسة البرمجيات والمستوى الرابع (مهيأ إدارياً) وفيه يراعى تفاصيل القياسات لعملية التطوير للمنتج مجمعة ومحددة كمياً بطريقة منظمة إدارياً. المستوى الخامس (الأمثل) وفيه تتم عمليات القياس باستمرار لتحسين أداء المنتج حيث يتم اقتراح طرق أخرى وسائل للاختيار بغرض التحسين.

إجابة التدريبات

تدريب (1)

الصحة (Correctness) ، الاعتمادية (Reliability).

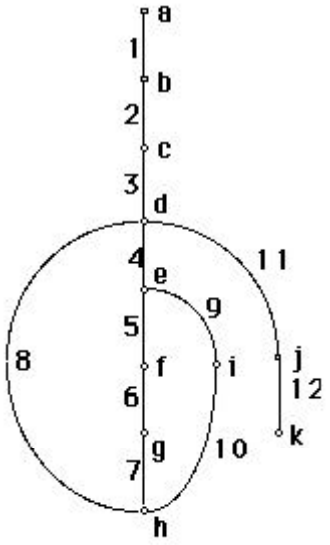
تدريب(2)

الصحة (Correctness) ، الاعتمادية (Reliability).

تدريب (3)

في خريطة تدفق التحكم الموضح بالشكل التالي :

- النقطة **(a)** ، والنقطة **(b)** تمثلان البداية والنهاية **(Start and Stop)**.
- النقطة (c) تمثل **GET A** ، والنقطة (i) تمثل **L = L +1** .
- الوصلات بين النقاط **(Arcs) : 5 , 6 & 9, 10** تمثل جملة الشرط **(IF-Else-Then)**.
- الوصلات بين النقاط **(Arcs) : 4 , 5 ,6, 7, 8** تمثل الحلقة **(While Loop)**.
- عدد النقاط **(Vertices) = N = 11** ، عدد الوصلات بين هذه النقاط **(Arcs) = E =12**.
- مدى تعقيد البرنامج = **E – N + 2 = 3** .



خريطة تدفق التحكم

Control Flow Graph

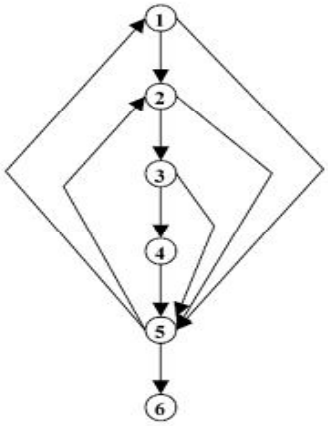
تدريب (4)

درجة تعقيد البرنامج **"P" = V(P) = E – N + 2**

ومن خلال خريطة التدفق يتضح أن عدد أسهم التوصيل "E" = 16 ، وعدد النقاط "N" = 13 ، وبذلك تكون V(P) = 5.

أو أن درجة تعقيد البرنامج "P" = V(P) = عدد نقاط اتخاذ القرار + 1 ، ومن خلال خريطة التدفق يتضح أن عدد نقاط اتخاذ القرار = 4 ، وبذلك تكون V(P) = 5 .

تدريب(5)



خريطة تدفق التحكم الخاصة بالإجرائية "Sort"

درجة تعقيد الإجرائية **"Sort" = V(Sort) = E – N + 2**

ومن خلال خريطة التدفق يتضح أن عدد أسهم التوصيل "E" = 10 ، وعدد النقاط "N" = 6 ، وبذلك تكون V(Sort) = 6.

أو أن درجة تعقيد الإجرائية ""Sort" = V(Sort) = عدد الحلقات المغلقة + 1 ، ومن خلال خريطة التدفق يتضح أن عدد الحلقات المغلقة = 5 ، وبذلك تكون(Sort)V = 6.

تدريب (6)

- **Unique operators : if ( ) { } > < = * ;**
- **Unique operands : k 2 3 x**

n1 = 10 , n2 = 4 , N1 = 13 , N2 = 7 , n = n1 + n2 = 14 , N = N1 + N2 = 20

$V = N \log_2 n = 20 \log_2 14 = 76.15$ Bit

مسرد المصطلحات

الصحة Correctness :

المدى التي تفي فيه البرمجية بمتطلبات المستخدم.

الإعتمادية (Reliability :

الحد الذي يمكن أن تعمل فيه البرمجية باستمرار دون حدوث توقف أو فشل.

الكفاءة Efficiency :

كمية RAM ووقت المعالج الذي تستخدمه البرمجية.

التكاملية Integrity :

الدرجة التي تمكن فيه البرمجية المستخدم من الدخول وتنظيم إدخال المعلومات.

الاستخدام Usability :

سهولة استخدام البرمجية.

الصيانة Maintainability :

الجهد المطلوب لاكتشاف خطأ وإصلاحه.

المرونة Flexibility :

الجهد اللازم لتغير البرمجية لمقابلة التغيرات المطلوبة.

الاختبارية Testability : الجهد اللازم لاختبار البرمجية بفعالية.

الانتقالية Portability :

الجهد اللازم لنقل البرمجية إلى حاسبات ذات مكونات وأنظمة تشغيل مختلفة.

إعادة الاستخدام Re-usability :

يمكن به إعادة استخدام البرمجية أو جزء منها من خلال برمجية أخرى.

تضافرية العمل Interoperability : الجهد اللازم لجعل البرمجية تعمل بالتعاون مع برمجية أخرى.

المصطلح بالإنجليزية	معناه بالعربية

المصطلح بالإنجليزية	معناه بالعربية
Activities	الأنشطة
Assembles	التجميع
Availability	مقياس للتوفر
Block of Sequential Code	كتلة من الشيفرة المتسلسلة
Capability Maturity Model (CMM)	نموذج نضج القدرة
Characteristics of Good Software Metrics	خصائص مقاييس البرمجيات الجيدة
Classification of Software Metrics	تصنيف مقاييس البرمجيات
Code Standards	المعايير القياسية للشيفرة
Collective Term	تعبير متراكم
Complexity Metrics	مقاييس التعقيد
Complexity of Delivered Product	تعقيد المنتج المسلم
Computed Metrics	المقاييس المحسوبة
Control Flow Graph	خريطة تدفق التحكم البيانية
Correctness	الصحة
Cost of Defects	تكلفة العيوب
Cost of Quality	تكلفة الجودة
Costs of Quality Activities	تكلفة أنشطة الجودة
Customer Satisfaction Index	دليل رضاء العميل
Cyclomatic Complexity Metric	مقياس تعقيد الانحناءات
Debugging performance	تنقيح البرنامج
Decide	يقرر
Decision Making	اتخاذ القرارات
Decision Point	تفرع أو نقطة قرار
Defect Ratios	نسب الأعطال

المصطلح بالإنجليزية	معناه بالعربية
Defect Removal Efficiency	كفاءة إزالة العيوب
Defects/KLOC	نسبة العيوب في ألف سطر شيفرة
Defined	معرف
Delivered Defect Quantities	كم الأعطال المستلمة
Demonstration	بيان عملي
Design or Program Inspection	فحوصات التصميم أو البرنامج
Design Standards	المعايير القياسية للتصميم
Detect Metric	مقاييس الكشف
Documentation	التوثيق
Documentation Standards	المعايير القياسية للتوثيق
Dynamic Metrics	المقاييس الديناميكية
Efficiency	الكفاءة
Established Criteria	معايير مُؤسسة
Evaluation of the Quality	تقويم الجودة
Fault Density "FD" Parameter	معامل قياس كثافة الأخطاء
Flexibility	المرونة
Formal Technical Review "FTR"	المراجعات التقنية الرسمية
Framework	إطار العمل
Halstead's Product Metric	مقاييس هالستيد للمنتج
IEEE	معهد المهندسين الكهربائيين والإلكترونيين
Initial	بدائي
Inspections	الفحوصات
Integrity	التكاملية
Interfaces	واجهات الاستخدام
Internal Code Documentation	التوثيق الداخلي للشيفرة

المصطلح بالإنجليزية	معناه بالعربية
Interoperability	تضافرية العمل
ISO Standard	مقياس آيزو المعياري
Legal Language Structures	الهياكل القانونية الصحيحة
Linearly Independent Paths	المسارات الخطية غير المعتمدة
Maintainability	قابلية الصيانة
Maintenance Effort	جهد الصيانة
Managed	مهيأ إداريا
Mathematical and Logical Operators	العوامل الحسابية والمنطقية
McCabe's cyclomatic Complexity	مقياس ماكابي لتعقيد الانحناءات
Mean Time Between Failures (MTBF)	مقياس المتوسط الزمني بين الأعطال
Metrics	المقاييس
Metrics and Quality Assurance of Software	مقاييس وتأمين جودة البرمجيات
Mean Time to Failure (MTTF)	مقياس المتوسط الزمني للعطل
Mean Time to Repair (MTTR)	مقياس المتوسط الزمني للتصليح
Objective	موضوعية
Organization 's Quality Policy	سياسية جودة المنظمة
Organization Standards	المعايير القياسية للمنظمة
Planned and Systematic Approach	منهجية مخططة ونظامية
Portability	الانتقالية
Prescribed Methods	طرق وصفية
Primitive Metrics	المقاييس البدائية
Process Metrics	مقاييس العملية
Processes	العمليات

المصطلح بالإنجليزية	معناه بالعربية
Processes Monitoring	مراقبة العمليات
Product Evaluation	تقويم المنتج
Product Metrics	مقاييس المنتج
Product Volatility	عدم ثبات المنتج
Programming Effort	الجهد المبذول في عملية البرمجة
Programming Process Efficacy Attribute	سمة مدى فعالية عملية البرمجية
Project Progress Reviews	مراجعات تقدم سير المشروع
Quality Assurance System	نظام تأمين الجودة
Quality Metric	مقياس الجودة
Quality Objectives	أهداف الجودة
Quality of Conformance	جودة التوافق
Quality of Design	جودة التصميم
Quality Plan	خطة الجودة
Quality Planning Activity	نشاط تخطيط الجودة
Quality Reviews	مراجعات الجودة
Quality Reviews Activity	نشاط مراجعات الجودة
Quantitative Aspects	السمات الكمية
Regulations	التنظيمات
Relevant Industry Standards	المعايير القياسية الصناعية وثيقة الصلة بالمشروع
Reliability	الاعتمادية
Reliability Metrics	مقاييس الموثوقية (الاعتمادية)
Repeatable	مثبت الجدارة
Reserved Words	الكلمات المحجوزة
Responsiveness	مدى الاستجابة

المصطلح بالإنجليزية	معناه بالعربية
Re-usability	إعادة الاستخدام
Reviews and Audits	المراجعات والتدقيقات
Re-work	إعادة العمل
Risks management	إدارة المخاطر
Robust	متين
Scope of Project Work	أفق عمل المشروع
Selection	الانتقاء
Software Engineering Institute "SEI"	معهد هندسة البرمجيات من جامعة ميلون بالولايات المتحدة
Software Metrics	مقاييس البرمجيات
Software Quality	جودة البرمجيات
Software Quality Assurance Activities	أنشطة تأمين جودة البرمجيات
Software Quality Management	إدارة جودة البرمجيات
Software Quality Management Activities	أنشطة إدارة جودة البرمجيات
Software Quality Management Importance	أهمية إدارة جودة البرمجيات
Software Resource Requirement	متطلبات موارد البرمجية
Specific Attributes	صفات معينة
Standards	المعايير القياسية
Standards Defined for the Project	المعايير القياسية المُعرفة للمشروع
Static Metrics	المقاييس الاستاتيكية
Storage Class Specifiers	مخصصات طبقة الذاكرة
Structured Program	البرامج الهيكلية
Style Conventions	اصطلاحات الأسلوب

المصطلح بالإنجليزية	معناه بالعربية
System Failure	فشل النظام
Test Coverage	تغطية الاختبار
Testability	الاختبارية
Total Number of Operands	عدد المعاملات الكلية في البرنامج
Total Number of Operators	عدد العوامل الكلية في البرنامج
Type Qualifiers	محددات النوع
Unique Operands	المعاملات المفردة
Unique Operators	العوامل المفردة
Usability	الاستخدام
Why Software Metrics	لماذا مقاييس البرمجيات؟

المراجع

أولاً : المراجع العربية :

[1] روجر بريسمان ، "هندسة البرمجيات" ، ترجمة مركز التعريب والترجمة بالدار العربية للعلوم ، الطبعة الأولى ، 2004م.

[2] مهندس عبد الحميد بسيوني ، "أساسيات هندسة البرمجيات" ، دار الكتب العلمية للنشر والتوزيع ، القاهرة ، 2005 م.

ثانياً : المراجع الإنجليزية :

[3] Ian Somerville , "Software Engineering", Addison Wesley, 2001.

[4] Ronald J. Leach, "Introduction to Software Engineering", CRC Press, 1999.

[5] Douglas Bell , "Software Engineering : A Programming Approach", 3rd Edition, Addison Wesley.

[6] Simon Alexandre, "Software Metrics An Overview (Version 1.0)" , Software Quality Lab , University of Namur, Belgium , July 2002.

[7] Fenton, N., and Neil, M. Software metrics: roadmap. In ICSE – Future of SE Track (2000), pp. 357–370.

[8] Goodman, P. Practical Implementation of Software Metrics. McGraw Hill, New-York, 1993.

[9] Ganesh Kandula (int04gka@cs.umu.se), "Product and Management Metrics for Requirements ", Master Thesis, Department of Computing Science , University Umeå , Umeå ,

Sweden , March 31st 2005.

[10] Sofia Nystedt, "Software Complexity and Project Performance" , Master thesis , Department of Informatics , School of Economics and Commercial Law , University of Gothenburg , 1999.

[11] [Grady92] Robert B. Grady, "Practical Software Metrics for project Management and Process Improvement", Prentice-Hall, Englewood Cliffs, 1992.

[12] Roger S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill, 1996.

[13] Arvind Gopu , "Software Metrics & Associated Issues A Literature Survey" , Graduate Student, Computer Science & Informatics , Indiana University, December 19, 2003.

[14] Everald E. Mills , " Software Metrics SEI Curriculum Module SEI-CM-12-1.1" , Software Engineering Department , Seattle University , Seattle, Washington 98122 , December 1988.

[15] Wikipedia, the free encyclopedia , Cyclomatic complexity , http://en.wikipedia.org/wiki/Software_metric

[16] Adam Garnett , "Software Metrics" , Version 2.3 , UNB, October , 2005.

ثالثاً : مواقع على شبكة الإنترنت تم الاستفادة منها :

[17] Rob Kremer , CPSC 451: Practical Software Engineering Complexity -- Software Metrics , http://sern.ucalgary.ca/courses/cpsc/451/W98/Complexity.html