عملية البرمجة:

يمكن وصف عملية البرمجة كما هو مبين في الشكل أدناه. حيث يوضع هيكل عام لعملية البرمجة common process framework وذلك بتعريف عدد صغير من نشاطات الهيكل التي يمكن تطبيقها على جميع المشاريع البرمجية بغض النظر عن حجمها أو تعقيدها، وعدد من مجموعات المهام حيث كل منها عبارة عن مجموعة من: مهام عمل هندسة البرامجيات، معالم مشروع، منتجات عمل برمجية ونواتج، ونقاط ضمان الجودة(التي تمكّن من تكييف نشاطات الهيكل مع خصائص المشروع البرمجي ومتطلبات فريق العمل في المشروع). وأخيراً تغلف نشاطات المظلة نموذج عملية البرمجة. إن نشاطات المظلة مستقلة عن أي نشاط للهيكل، وهي حدث يجري طوال عملية البرمجة.

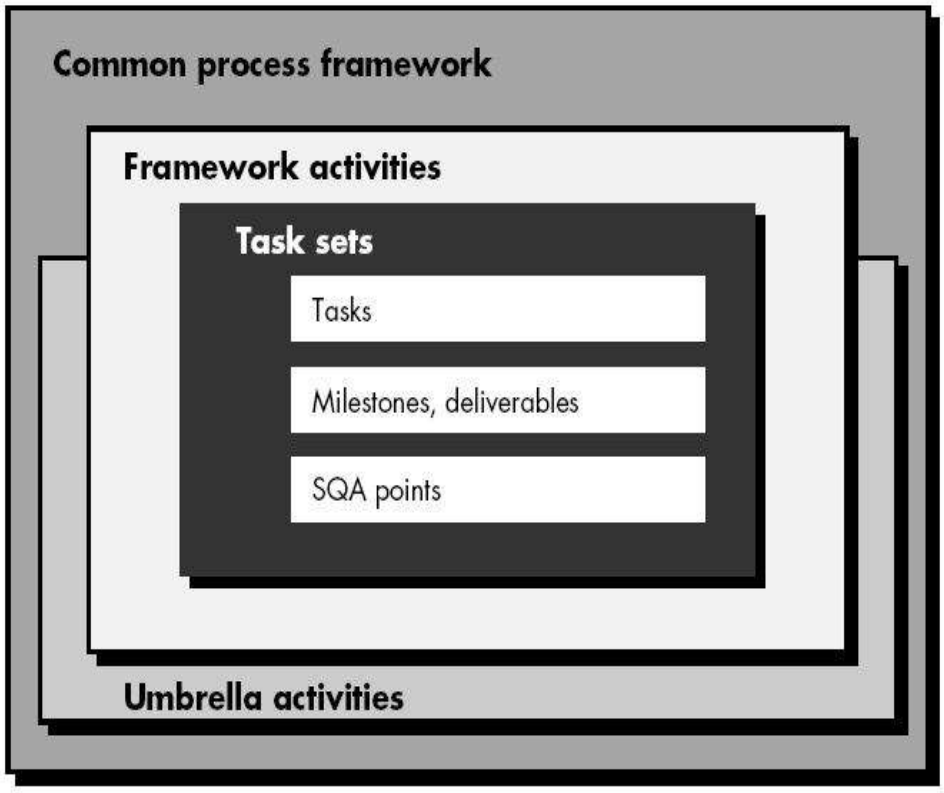


FIGURE
The software process

ة

البرامجيات SEI نموذجا شاملا يستند إلى مجموعة من مقدرات هندسة البرامجيات التي يجب أن تكون مكتسبة مع وصول المؤسسات إلى مستويات مختلفة من نضج عملية البرمجة. ولتحديد حالة

المؤسسة الراهنة لنضج عملية البرمجة يستعمل معهد الـ SEI استمارة تقييم وسلم تصحيح ذي خمسة مستويات. يحدد سلم التصحيح هذا مدى التوافق مع نموذج نضج القدرات CMM (Capability Maturity Model) (الذي يعرف نشاطات أساسية مطلوبة على مستويات مختلفة من نضج عملية البرمجة. يرسم منهج معهد الـ SEI خمسة مستويات لنضج عملية البرمجة تعرّف على النحو التالي:

المستوى ١: ابتدائي– توصف عملية البرمجة بأنها منسابة ad hoc وأحيانا فوضوية chaotic . عمليات برمجة قليلة فقط هي المحدد جيداً، ويعتمد نجاحها على المجهود الفردي.

المستوى ٢: متكرر – توضع عمليات برمجة أساسية لإدارة المشروع وذلك لمتابعة الكلفة والجدول الزمني والوظائفية، ويكون نظام عملية البرمجة الضروري جاهزا لتكرار النجاحات السابقة التي حصلت في المشاريع ذات تطبيقات مشابهة.

المستوى ٣: معرّف – تكون عملية البرمجة للنشاطات الإدارية الهندسية موثقة وقياسية ومتكاملة مع عملية البرمجة لكل المؤسسة. تستخدم جميع المشاريع نسخة مصدقة وموثقة من عملية المؤسسة لتطوير وصيانة البرامجيات. يتضمن هذا المستوى جميع الخصائص المعّرفة للمستوى ٢ .

المستوى ٤: مُدار – يجمع قياسات تفصيلية لعملية البرمجة وجودة المنتج. وتفهم كل من عملية البرمجة والمنتجات كمّياً، ويتحكم بهما باستخدام قياسات تفصيلية. يتضمن هذا المستوى جميع الخصائص المعّرفة للمستوى ٣.

المستوى ٥: مثالي – تتحقق تحسينات مستمرة على عملية البرمجة بتكرار كمية منها، ومن اختبار أفكار وتكنولوجيات مبتكرة. يتضمن هذا المستوى جميع الخصائص المعّرفة للمستوى ٤.

نماذج عملية البرمجة :

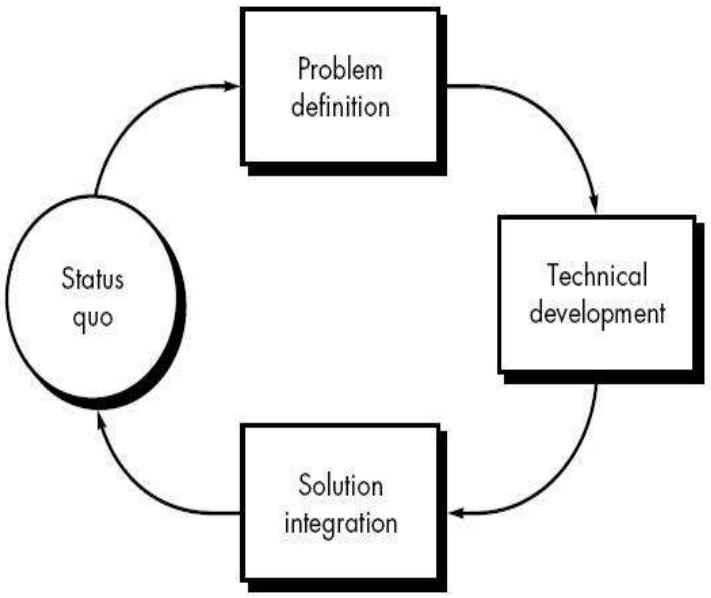
لكي يتمكن مهندس البرمجيات أو فريق المهندسين من حل مسائل حقيقية في بيئة صناعية، عليهم استخدام إستراتيجية تطوير تشمل(عملية البرمجة، الطرق وطبقات الأدوات المشروحة سابقا، والمراحل العامة التي ناقشناها، عادة تسمى هذه الإستراتيجية (نموذج عملية البرمجة) Process mode أو (نموذج هندسة البرمجيات). يتم اختيار نموذج عملية هندسة البرمجيات بناءً على طبيعة المشروع أو التطبيقات، الطرق والأدوات المستخدمة، والتحكم والأداء المطلوبين.وقد استخدم L.B.S. [RAC95] Raccoon في مقالته العلمية المثيرة للاهتمام حول طبيعة عملية البرمجة : Fractals (المتشابهات ذاتياً أو المتجزّئات) أساساً لمناقشة الطبيعة الحقيقية لعملية البرمجة.

يمكن وصف كل التطوير البرمجي بحلقة لحل أي مشكلة (الشكل أدناه) حيث تواجه أربع مراحل مختلفة هي : الوضع الراهن (Status quo)، وتعريف المشكلة، والتطوير التقني، ومكاملة الحل .

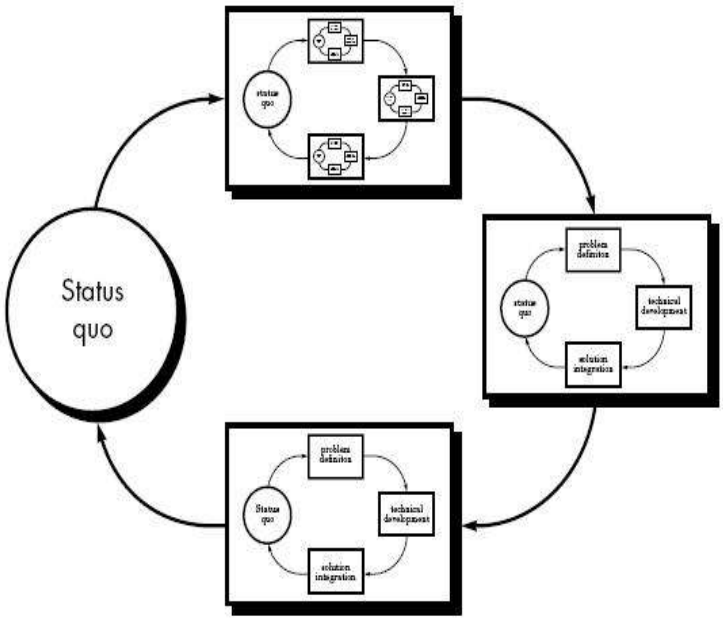
FIGURE

(a) The phases of a problem solving loop [RAC95]

(b) The phases within phases of the problem solving loop [RAC95]



(a)



(b)

المسخه المحدده المطلوب حلها، في حين إن التطوير التقني يحل المسخه عن طريق تطبيق تكنولوجيا ما، ومكاملة الحل تقدم النتائج (كوثائق، برامج، بيانات، وظائف الإعمال الجيدة، منتج جديد) لأولئك الذين طلبوا الحل أول مرة يمكن إسقاط الخطوات والمراحل العامة لهندسة البرمجيات بسهولة على هذه المراحل.

تنطبق حلقة حل المشكلة الموصوفة سابقاً على إعمال هندسة البرمجيات عند عدة مستويات مختلفة، ويمكن استخدامها عند المستوى الماكروي (Macro Level) عند النظر إلى كامل التطبيق أو البرنامج، وعند المستوى المتوسط عندما تتم هندسة مكونات البرنامج، أو حتى عند مستوى كتابة شيفرة البرنامج. لذا يمكن استخدام تمثيل المتجزئات (Fractals)[1] ، لتقديم رؤية مثالية لعملية البرمجة، تتضمن كل مرحلة في حلقة حل المشكلة وهكذا (يستمر هذا إلى حد معقول، في البرمجيات : سطر من الشيفرة).

يصعب واقعياً تقسيم النشاطات إلى أجزاء مستقلة تقسيماً أنيقا كما يوحي الشكل أعلاه (a) وذلك بسبب وجود تداخل ضمن المراحل وفيما بينها، ومع ذلك تقود هذه الرؤية المبسطة إلى فكرة هامة جداً وهي انه : بغض النظر عن نموذج عملية البرمجة المختارة للمشروع البرمجي فان جميع المراحل (الوضع الراهن، تعريف المشكلة، التطوير التقني، مكاملة الحل) تتواجد معاً بمستوى معين من التفصيل، وبملاحظة الطبيعة التكرارية للشكل أعلاه (b) فان المراحل الأربع المناقشة سابقاً تنطبق بنفس القدر على تحليل تطبيق أو برنامج كامل أو على توليد مقاطع صغيرة من الشيفرة.

يقترح Raccoon [RAC95] نموذجاً فوضوياً (chaos model) وهو يصف تطوير البرمجيات كاستمرار من المستخدم إلى المطور إلى التكنولوجيا. ومع تقدم العمل نحو إنشاء نظام متكامل، يتم تطبيق المراحل الموصوفة سابقاً تطبيقاً متكرراً وفقاً لمتطلبات المستخدم والمواصفات التقنية للبرمجيات التي يضعها المطور.

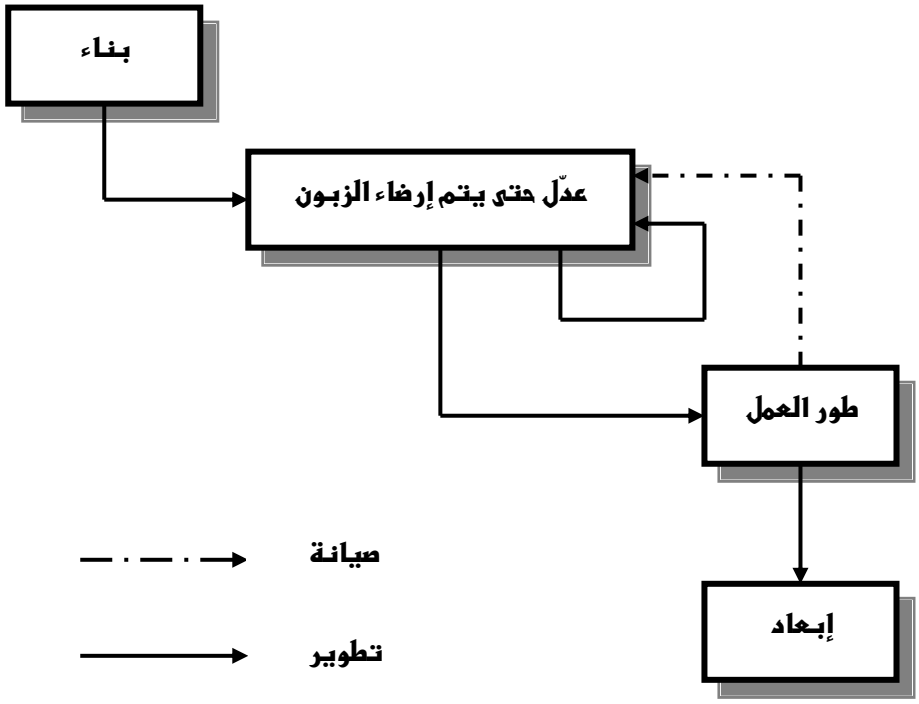
ستناقش الفقرات القادمة نماذجاً مختلفة للعمليات برمجة في هندسة البرمجيات يمثل كل نموذج منها محاولة لإضفاء ترتيب وتنظيم على نشاط فوضوي (غير منتظم). ومن المهم أن نتذكر إننا قد شرحنا كل نموذج من النماذج بطريقة تساعد على التحكم والتنسيق في المشاريع البرمجية الحقيقية، ومع ذلك يبدي جميع النماذج بعض خصائص النموذج الفوضوي.

١. بناء ثم إصلاح Build-and-Fix:

من المؤسف أن يكون هنالك العديد من المنتجات قد طورت باستخدام ما يسمى طريقة بناء– ثم–إصلاح. حيث إن المنتج الذي بني دون محددات أو أي محاولة للتصميم. بدل ذلك، فان المطورين بكل بساطة يقومون ببناء المنتج ويعيدون العمل عليه بقدر عدد المرات الضرورية لإرضاء الزبون. وهذه الطريقة موضحة في الشكل التالي:

---

[1]) اقترحت المتجزئات أصلا للتمثيلات الهندسية، وذلك بان يعرف نمط (Pattern) ثم يطبق تكرارياً على نطاقات اصغر تطبيقاً متتالياً بحيث تقع الأنماط ضمن بعضها بعض.



شكل يوضح نموذج بناء–ثم–إصلاح

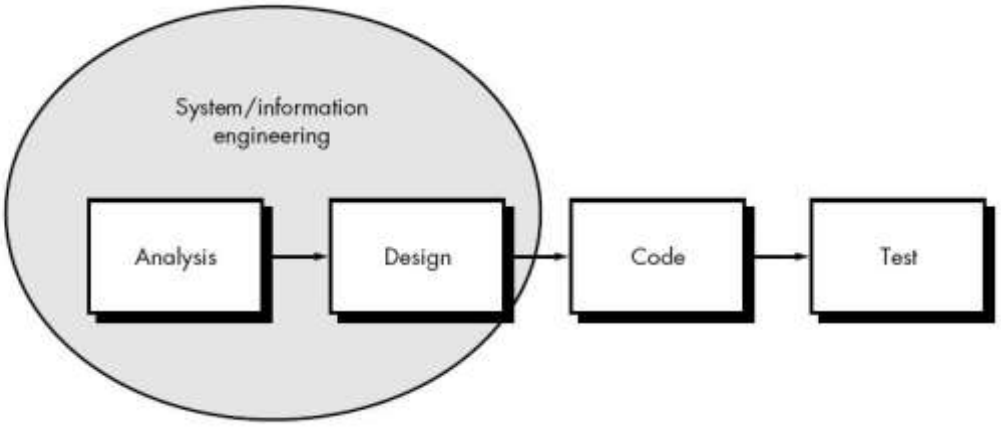
تعمل هذه الطريقة بصورة جيدة على تمارين البرمجة ذات طول يتراوح من ١٠٠ إلى ٢٠٠ سطر. إن هذا الموديل بصورة عامة لا يستخدم لبناء منتج، مهما كان حجمه معقول. حيث إن كلفة التغيرات على منتج برمجي تكون صغيرة إذا ارتبطت مع طور التحليل، تحديد متطلبات، أو التصميم. ولكنها تكبر و تتنامى بصورة غير مقبولة إذا حدثت التغيرات بعد وضع شيفرة البرنامج، وتكون أسوء إذا كان البرنامج في طور العمل. لذالك فان كلفة طريقة بناء–ثم–إصلاح هي في الواقع اكبر بكثير من كلفة المنتجات المصممة جيدا أو الموضوع محددات لها بصورة مناسبة. بالإضافة إلى ذلك فان الصيانة المنتج ممكن أن تكون صعبة جداً من دون توثيقات وضع المحددات والتصميم، وفرص الوقوع في الأخطاء ابر بصورة ملحوظة.

٣. النموذج ألتتابعي الخطي :

يوضح الشكل أدناه النموذج ألتتابعي الخطي (Linear sequential) لهندسة البرمجيات يقترح هذا النموذج الذي يسمى أحياناً "دورة الحياة التقليدية"أو"النموذج ألشلالي"(waterfall

model) منهجاً تتابعياً منتظماً[1] .لتطوير البرمجيات، يبدأ عند مستوى النظام ويتقدم تباعاً إلى التحليل فالتصميم فالتشفير فالاختبار فالصيانة.

FIGURE
The linear sequential model



يشتمل النموذج ألتتابعي الخطي، الذي جرت نمذجته على شاكلة الدورة الهندسية المألوفة، على النشاطات التالية:

أ. **هندسة ونمذجة النظام / المعلومات** : نظراً إلى إن البرنامج هو دوماً جزء من النظام أو (إعمال) اكبر فان العمل يبدأ بوضع متطلبات لجميع مكونات النظام ، ثم تخصيص مجموعة جزئية من هذه المتطلبات للبرنامج وهذه الرؤيا للنظام أساسية عندما يجب ربط البرنامج بالمكونات الأخرى، مثل العتاد والأشخاص وقواعد البيانات . تشمل هندسة النظام وتحليله على تجميع المتطلبات عند مستوى النظام، مع قدر بسيط من التحليل والتصميم عند المستوى الأعلى . أما هندسة البرنامج فتشتمل على تجميع المتطلبات عند المستوى الاستراتيجي للأعمال وعند مستوى مجال الأعمال.

ب. **تحليل متطلبات البرنامج** : تتوقف عملية تجميع المتطلبات وتركز على البرنامج بوجه خاص، وحتى يفهم المهندس (المحلل) طبيعة البرنامج أو (البرنامج) المطلوب بناؤه، يجب عليه فهم نطاق معلومات البرنامج، إضافة إلى الوظيفة أو السلوك والأداء والواجهات المطلوبة، ويجب أن يجري توثيق متطلبات النظام والبرنامج ومراجعتها مع الزبون.

ج. **التصميم** : تصميم البرمجيات هو عملية برمجة متعددة الخطوات تهتم بأربع مميزات للبرنامج :- (بنية البيانات، بنيان البرمجية، تمثيلات الواجهة، وتفاصيل عملية البرمجة (الخوارزمية)) تقوم عملية التصميم بترجمة المتطلبات إلى تمثيل البرمجيات يمكن تقييمه من

---

١) مع إن النموذج ألشلالي الأصلي الذي اقترحه Winston Royce [ROY70] يوفر حلقات تغذية رجعية، إلا أن غالبية المؤسسات التي تطبق هذا النموذج لعملية البرمجة تعامله وكأنه خطي حصراً.

حيث الجودة قبل البدء بتوليد الشفرة وكما جرى للمتطلبات يتم توثيق التصميم بحيث يصبح هذا التوثيق جزءً من تشكيلة البرنامج .

د. **توليد الشفرة** : يجب ترجمة التصميم إلى صيغة تستطيع الآلة (الكومبيوتر) قرأتها وتقوم بهذه المهمة مرحلة توليد الشفرة وإذا نفذ التصميم بطريقة مفصلة يمكن عندها تحقيق توليد الشفرة ميكانيكياً .

هـ. **الاختبار** : يبدأ اختبار البرنامج حال توليد الشفرة وتركز عملية الاختبار على منطقية العلاقات الداخلية للبرنامج بحيث تضمن اختبار جميع العبارات وعند مستوى العلاقات الخارجية أي تنفيذ اختبارات لكشف الأخطاء ولضمان أن الدخل المعرف سيعطي نتائج فعلية تتوافق مع النتائج المطلوبة.

و. **الصيانة** : تخضع البرمجيات بلا شك لتعديلات بعد تسليمها للزبون (الاستثناء المحتمل هو برمجيات الأجهزة،embedded software)، يتم التغيير بسبب الأخطاء التي جرت مواجهتها، أو لأنه يجب تكييف البرمجيات لتستوعب التغييرات في بيئتها الخارجية (مثال: يطلب تغيير بسبب شراء نظام تشغيل جديد أو جهاز جديد)، أو لان الزبون طلب تنفيذ تحسينات على الأداء أو الوظيفة عند تنفيذ الصيانة، ستطبق كل مرحلة من المراحل السابقة على البرنامج الموجود عوضاً عن تطبيقها على برنامج جديد.

إن النموذج ألتتابعي الخطي هو النموذج الأقدم والأكثر استخداماً لهندسة البرمجيات، إلا إن النقد الذي تم التعرض له في البدء أدى إلى التشكيك في فعاليته، حتى من داعميه النشطين [HAN95]، من المشاكل التي تظهر أحيانا عند تطبيق النموذج ألتتابعي الخطي هي :

١. نادراً ما تتبع المشاريع الحقيقية التقدم ألتتابعي الذي يقترحه النموذج ، ومع إن النموذج الخطي يمكن أن يستوعب التكرار، إلا انه يفعل ذلك بأسلوب غير مباشر، ونتيجة لذلك، قد تسبب التعديلات ارتباكا (فوضى) مع تقدم فريق المشروع.

٢. يصعب غالبا على الزبون طرح جميع متطلباته بوضوح، والنموذج الخطي يحتاج إلى ذلك، ولهذا تبرز صعوبات في استيعاب عدم التحديد الطبيعي للمتطلبات الذي يتم في العديد من المشاريع.

٣. يجب أن يتحلى الزبون بالصبر، إذ لن تتوفر نسخة عاملة من البرنامج أو (البرامج) حتى وقت متأخر من الجدول الزمني للمشروع، وقد يكون الخطأ الفادح كارثيا إذا لم يكشف إلا عند مراجعة البرنامج العامل (المنتج النهائي).

٤. غالبا ما يتأخر المطورون لأسباب غير ضرورية ، فقد وجد Bradac [BRA94] في تحليل مثير للاهتمام إن الطبيعة الخطية لدورة الحياة التقليدية تقود إلى "حالات انتظار"(blocking states) يضطر فيها بعض أعضاء فريق المشروع أشخاص آخرين من الفريق لإنهاء مهام مترابطة (غير مستقلة)، وفي الواقع قد يتجاوز وقت الانتظار هذا الزمن المصروف على إعمال منتجة هناك مثل كان تحصل حالات انتظار بكميات اكبر في بداية ونهاية عملية البرمجة التتابعية الخطية.

ومع إن كل هذه المشاكل هي مشاكل حقيقية، إلا إن لنموذج دورة الحياة التقليدية مكانا محددا وهاما في عمل هندسة البرمجيات، فهو يزودنا بقالب توضع فيه طرق للتحليل والتصميم والتشفير والاختبار والصيانة ولا تزال دورة الحياة التقليدية لهندسة البرمجيات هي نموذج عملية البرمجة الأكثر استخداماً.وبالرغم من وجود نقاط ضعف فيها، إلا أنها أفضل بكثير من تطوير البرمجيات وفق منهج المصادفة.

٣. نموذج "النمذجة الأولية":

غالبا ما يعرف الزبون مجموعة من الأهداف العامة للبرنامج، ولا يحدد بالتفصيل كل متطلبات الدخل أو المعالجة أو الخرج، وقد يكون المطور في حالات أخرى غير متأكد من فعالية الخوارزمية، أو من تكيف نظام التشغيل، أو الشكل الذي يجب أن يأخذه تفاعل الإنسان مع الآلة، لذلك فان نموذج النمذجة الأولية (Prototyping Paradigm) يقدم الطريقة الفضلى في هذه الحالات وحالات كثيرة غيرها.

تبدأ النمذجة الأولية (الشكل أدناه) بجمع المتطلبات لذلك يجتمع المطور والزبون لتعريف الأهداف الإجمالية للبرنامج، وتحديد أية متطلبات معروفة، وتحديد عناوين المجالات التي تتطلب تعريفات أكثر ويوضع عندئذ "تصميم سريع" يركز التصميم السريع على تمثيل نواح محددة من البرنامج، وخاصة تلك التي ستكون مرئية للزبون أو المستخدم (كطرق الإدخال وصيغ الإخراج) يقود التصميم السريع إلى نموذج أولي (Prototype) ، يعيد الزبون أو المستخدم تقييم النموذج الأولي ويستخدم ذلك التقييم لتصفية متطلبات البرنامج التي يجب تطويرها ويحدث تكرار من خلال ضبط النموذج الأولي لتحقيق متطلبات الزبون، وهذا يمكّن المطور في الوقت ذاته من إيجاد فهم أفضل للحاجات الواجب تلبيتها.

مثالياً يستخدم النموذج الأولي كآلية لتحديد متطلبات البرنامج وبناء نموذج أولي ويحاول المطور الاستفادة من أجزاء البرنامج الموجودة أو تطبيق أدوات (مثل مولدات التقارير، إدارة الأطر، ...الخ)تمكنه من توليد برامج عاملة بسرعة.

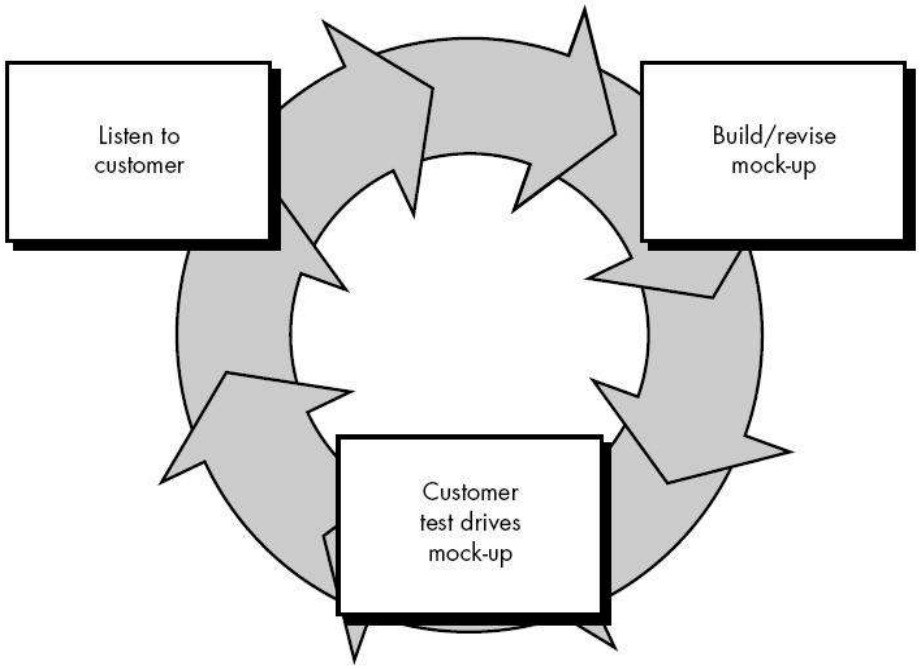


FIGURE
The prototyping paradigm

ولكن ماذا تفعل بالنموذج الأولي بعد انتهاء الأهداف المحددة سابقاً ؟ يقدم [BRO75] Brooks جوابا عن ذلك :

في معظم المشاريع، بالكاد يكون أول نظام يبنى قابلا للاستخدام فقد يكون بطيئا جدا، أو كبيرا جدا، أو صعب الاستخدام ، أو الثلاثة معا. فلا بديل عن البدء من جديد وبناء نسخة أعيد تصميمها بحيث تحل هذه المشاكل. . . ويجب عند استخدامك مفهوم نظام جديد أو تقنية جديدة أن تبني نظاما وأنت على علم بأنه سيرمى جانبا، ذلك لأنه حتى مع وجود أفضل تخطيط لن يكون ملما بكل شيء إلى درجة بناء نظام صحيح من المرة الأولى، لذلك فان السؤال الإداري هو: هل نبني نظاما رائداً ونرميه جانبا ؟ عمليا سوف نفعل ذلك. ويبقى السؤال الوحيد: هل التخطيط المسبق هو لبناء نظام ثم لرميه؟ أو للوعد بتسليم هذه النسخة (التي سترمى) إلى الزبون؟

يمكن استعمال النموذج الأولي كـ "النظام الأول" (the first system) الذي ينصح Brooks برميه، ولكن هذا الرأي قد يكون مثاليا، صحيح إن كلا من المطور والزبون يحب نموذج النمذجة الأولية، إذ يتمكن الزبون من اخذ فكرة عن النظام الفعلي، ويستطيع المطورون بعد ذلك البدء فورا بتنفيذ شيء ما، ولكن النمذجة الأولية قد تكون مليئة بالمشاكل للأسباب التالية:

١. يرى الزبون في النموذج الأولي ما يبدو انه نسخة صحيحة (عاملة) من البرنامج وهو غير مدرك إن هذا النموذج الأولي قد حوفظ عليه مترابطا ترابطا واهيا جدا، ولا يدرك إننا بسبب السرعة في انجازه لم نأخذ بالحسبان الجودة الكلية للبرنامج أو لقابلية صيانته(maintainability) على المدى الطويل. وعندما يتم إبلاغ الزبون بوجوب إعادة بناء المنتج بحيث يراعى تحقيق معايير عالية الجودة فانه يصرخ كالمجنون ويطلب إجراء "بعض الإصلاحات" لجعل النموذج الأولي منتجا يعمل جيدا، وغالبا ما تستجيب إدارة تطوير البرمجيات لهذا الطلب.

٢. غالبا ما يجري المطور بعض التجاوزات في الانجاز (implementation) لجعل النموذج الأولي يعمل بسرعة. فقد يستخدم نظام تشغيل أو لغة برمجة غير مناسبين، فقط لأنهما متوافران ومعروفان، وقد يعتمد خوارزمية غير كافية، فقط لإيضاح الإمكانيات. وقد يصبح المطور بعد مدة أكثر معرفة والماماً بهذه الخيارات وينسى كل أسباب كونها غير مناسبة، ويصبح الآن الخيار الأقل مثالية جزءاً متكاملاً من النظام.

بالرغم من إمكانية حصول مشاكل إلا إن استخدام النموذج الأولي يمكن أن يكون منهجا فعالا لهندسة البرمجيات والسر هو تحديد قواعد اللعبة منذ البداية، أي ، يجب أن يتفق المطور والزبون على إن النموذج الأولي يبنى لكي يستخدم كآلية لتعريف المتطلبات، ثم يهمل (على الأقل جزئيا) وتبنى البرمجيات الفعلية مع مراعاة الجودة والصيانة.

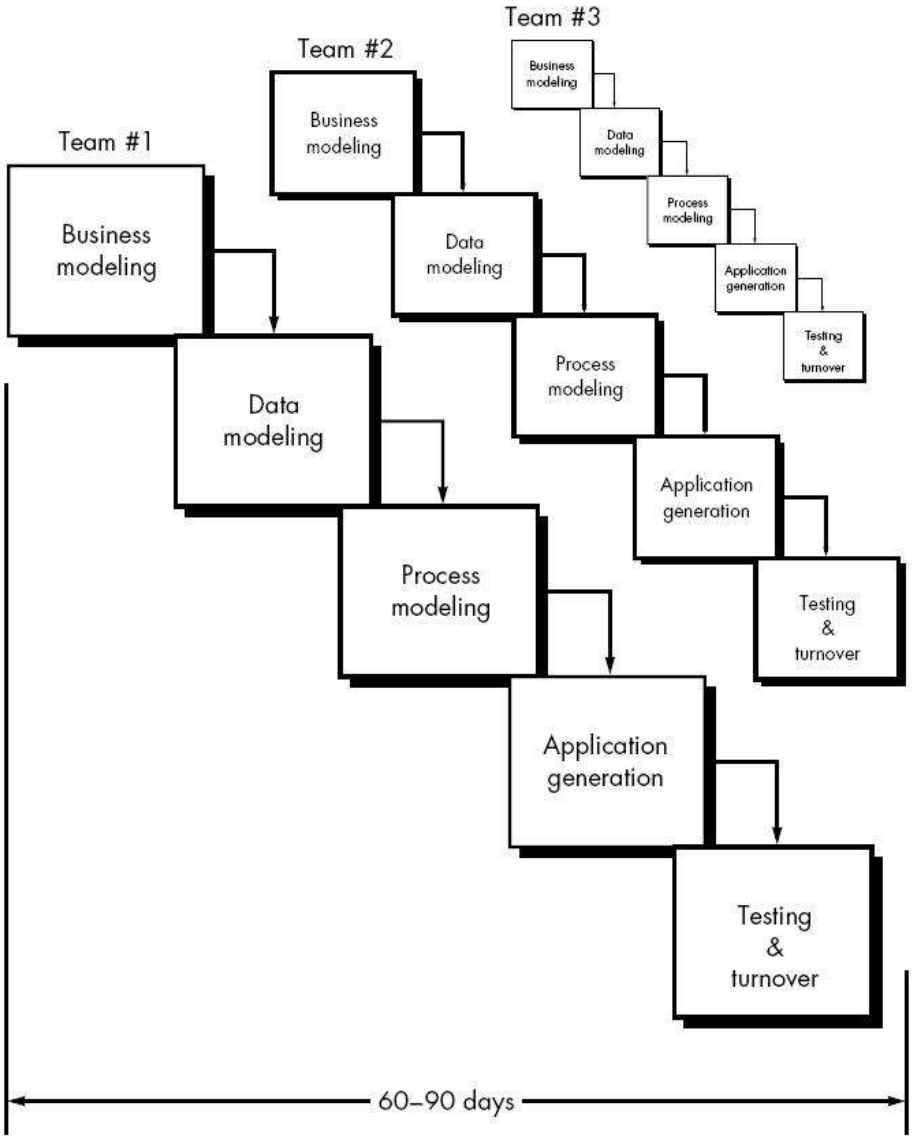
٤. نموذج التطوير السريع للبرنامج :

التطوير السريع للبرنامج (RAD) أو(Rapid Application Development) هو نموذج تتابعي خطي لعملية تطوير البرمجيات يهتم بدورة تطوير قصيرة جدا، النموذج RAD هو تكييف "سريع جدا" للنموذج ألتتابعي الخطي، حيث يجري تطوير سريع باستخدام أسلوب بناء يعتمد على المكونات إذا فهمت المتطلبات جيدا وحصرت أفق المشروع[١] ، فان عملية البرمجة RAD تسمح لفريق التطوير بإيجاد "نظام يعمل تماماً" (fully functional system) خلال مدة قصيرة جدا (من ٦٠ إلى ١٢٠ يوما) [MAK91] يتضمن الأسلوب RAD، الذي تم استخدامه استخداما أوليا في تطبيقات أنظمة المعلومات، المراحل التالية [KER94] .

---

١) هذه الشروط مضمونة في الحقيقة، كثير من المشاريع البرمجية حددت المتطلبات في البداية تحديدا سيئا جدا، وفي حالات كهذه يكون استخدام النموذج الأولي(الفقرة٢-٥) أو الطرق التطويرية (الفقرة ٢-٧) خيارا أفضل بكثير لعملية البرمجة [REI95] .

FIGURE
The RAD model



نمذجة الأعمال (business) : ينمذج تدفق المعلومات بين وظائف الأعمال بطريقة تجيب عن الأسئلة التالية: ما المعلومات التي تقود (تسير) عملية برمجة الأعمال؟ ما المعلومات التي يجري توليدها؟ من الذي يولدها؟ أين تذهب المعلومات؟ من يعالجها؟

نمذجة البيانات : نلخص أولا تدفق المعلومات، الذي عرف انه جزء من مرحلة نمذجة الأعمال وتدفقه، في مجموعة من أهداف البيانات اللازمة لدعم الأعمال ثم نحدد المميزات (تسمى سمات، attributes) لكل هدف ويتم تعريف العلاقات بين هذه الأهداف.

نمذجة عملية البرمجة : تحول أهداف البيانات، التي تم تعريفها في مرحلة نمذجة البيانات لتحقيق تدفق المعلومات الضروري، اللازمة بدورها لتحقيق وظيفة الأعمال، ثم نعمل على توليد سمات معالجة لكل عملية من عمليات إضافة أو تعديل أو حذف أو استرجاع أهداف البيانات.

توليد التطبيق: يفترض التطوير السريع للبرنامج RAD استخدم تكنولوجيات الجيل الرابع 4GT (Fourth Generation Technology) . فبدلا من إيجاد برمجيات باستخدام لغات برمجة تقليدية من الجيل الثالث، تعمل عملية التطوير السريع للبرنامج RAD على إعادة استخدام مكونات البرنامج الموجودة(عندما يكون ممكنا)أو إنشاء مكونات قابلة لإعادة الاستخدام (عند الضرورة) وتستخدم الأدوات المؤتمتة في كل الأحوال لتسهيل بناء البرنامج.

الاختبار والتطوير(Turnover) : لما كان التطوير السريع للبرنامج RAD يشدد على إعادة الاستخدام، سيكون قد سبق وتم اختبار العديد من مكونات البرنامج، وهذا ما يقلل من زمن الاختبار الكلي ومع ذلك يجب اختبار المكونات الجديدة وتجربة كل الواجهات تجريبا كاملا.

إن نموذج عملية البرمجة RAD الموضح في الشكل السابق يوضح إن التقييد الزمني المفروض على مشروع RAD يتطلب أفقا قابلا للتحجيم (scalable scope) [KER94] .فإذا أمكن تقسيم تطبيق الأعمال بحيث نستطيع انجاز كل وظيفة رئيسة في اقل من ثلاثة أشهر (باستخدام المنهج الموصوف سابقا) يكون هذا التطبيق مرشحا جيدا لـ RAD ، يمكن تناول كل وظيفة رئيسة بواسطة فريق RAD مستقل، ثم تتكامل لتشكل كيانا واحداً.

الأسلوب RAD ككل نماذج عملية البرمجة له بعض المساوئ [BUT94]:

- يتطلب RAD في حالة المشاريع الكبيرة القابلة للتحجيم موارد بشرية كافية لإنشاء العدد المطلوب من الفرق RAD ،
- يتطلب RAD مطورين وزبائن ملتزمين بالنشاطات السريعة والمتلاحقة الضرورية لإتمام النظام في زمن مختصر جدا، فإذا فقد الالتزام من احد الطرفين، ستخفق مشاريع RAD.

ليست كل أنماط التطبيقات مناسبة لاستعمال RAD فان لم يكن ممكنا تقسيم النظام بشكل مناسب ستبرز مشاكل عند بناء المكونات الضرورية لـRAD ، وكذلك إذا كان الأداء العالي هو احد المتطلبات وكان علينا تحقيقه بتوليف الواجهات مع مكونات النظام، فقد لا يعمل الأسلوب RAD من جهة أخرى، RAD غير مناسب أيضا عندما تكون المخاطر التقنية عالية، ويحدث هذا عندما يستخدم تطبيق جديد تكنولوجيا جديدة استخداما كثيفا أو عندما تتطلب البرمجيات الجديدة درجة عالية من التشغيلية البينية (interoperability أو قابلية العمل المتبادل) لبرامج الكمبيوتر.

يركز نموذج التطوير السريع للبرنامج RAD على إيجاد مكونات برامج قابلة لإعادة استخدام فإعادة الاستخدام هي حجر الزاوية للتكنولوجيات الغرضية وسنصادف مبدأ إعادة الاستخدام أيضا في نموذج عملية تجميع المكونات.

٥. النماذج التطورية لعملية البرمجة :

هناك اعتراف متزايد إن البرمجيات، مثل جميع الأنظمة المعقدة، تتطور على مر الزمن [GII88] إذ تتغير غالبا متطلبات الأعمال والمنتج مع تقدم عملية تطوير هذا المنتج، وهذا ما يجعل المسار المستقيم باتجاه إنتاجه غير واقع، يضاف إلى ذلك إن المواعيد المقيدة لطرح المنتج في السوق تجعل إكمال منتج برمجي شامل شيئا مستحيلا، ولكن عندما نرغب بتقديم نسخة محدودة لمواجهة المنافسة أو ضغط العمل نجد هذا النموذج جيدا شرط أن تكون نواة المنتج أو متطلبات النظام مفهومة جدا، ولو لم تعرف بعد توسيع المنتج أو النظام، يحتاج مهندسو البرمجيات في هذه الحالات وفي حالات أخرى مشابهة إلى نموذج لعملية البرمجة صمم بوضوح (بالتفصيل) لاستيعاب منتج يتطور مع الزمن.

لقد صمم النموذج ألتتابعي الخطي لحالات التطوير المباشر (خط مستقيم) وبمعنى آخر، تفترض هذه الطريقة الشلالية انه سيتم تسليم كامل النظام بعد اكتمال هذا التتابع الخطي، ومن جهة أخرى فقد صمم النموذج الأولي لمساعدة الزبون أو (المطور) على فهم المتطلبات، ولم يصمم عموما لتسليم نظام نهائي. لم تلاحظ الطبيعة التطورية للبرمجيات في كلا هذين النموذجين الاتباعيين (التقليديين) لهندسة البرمجيات.

إن النماذج التطورية (Evolutionary models) تكرارية، وتوصف بطريقة تمكن مهندس البرمجيات من تطوير نسخ أكثر تعقيدا من البرمجيات، وسنذكر فيما يلي أربعة منها.

٥-١. النموذج ألتزايدي :

يجمع النموذج ألتزايدي (incremental mode) عناصر من النموذج ألتتابعي الخطي (مطبقة بصورة متكررة) مع الفلسفة التكرارية. وعليه يقوم النموذج ألتزايدي (كما في الشكل الاحق) بتطبيق تتابعات خطية بأسلوب متعاقب مع تقدم زمن الإنتاج (الجداول الزمنية للإنتاج) وينتج كل تتابع خطي تزايدا من البرنامج يمكن تسليمه للزبون [MCDE94] فمثلا قد تقدم برمجيات معالجة النصوص المطورة باستخدام النموذج ألتزايدي إدارة أساسية للملفات والتحرير ووظائف إنتاج الوثائق في التزايد الأول، وتقدم إمكانيات أكثر تطورا (تعقيدا وتقدما) كتحرير وإنتاج وثائق في التزايد الثاني، وتقدم التصحيح الإملائي والقواعد في التزايد الثالث، وتقدم إمكانات متقدمة لضبط تنسيق الصفحة (page layout) في التزايد الرابع، ومن الجدير بالذكر انه يمكن استخدام نموذج النمذجة الأولية في سير عملية البرمجة في أي تزايد.

يكون التزايد الأول عادة عبارة عن نواة منتج (core product) عند استخدام النموذج ألتزايدي، أي انه يتم تحديد المتطلبات الأساسية، ولكن تبقى مظاهر إضافية عديدة غير محققة (بعضها معروف والآخر غير معروف). يستعمل المستخدم نواة المنتج (أو تخضع هذه النواة لمراجعة تفصيلية)، ويجري نتيجة للاستخدام و/أو للتقييم تطوير خطة التزايد التالي، وتحدد الخطة تعديلات نواة المنتج لتلبية حاجات الزبون بشكل أفضل وتقديم مزايا ووظائف إضافية، وتتكرر هذه العملية بعد تسليم كل تزايد حتى يتم إنتاج كامل النظام.

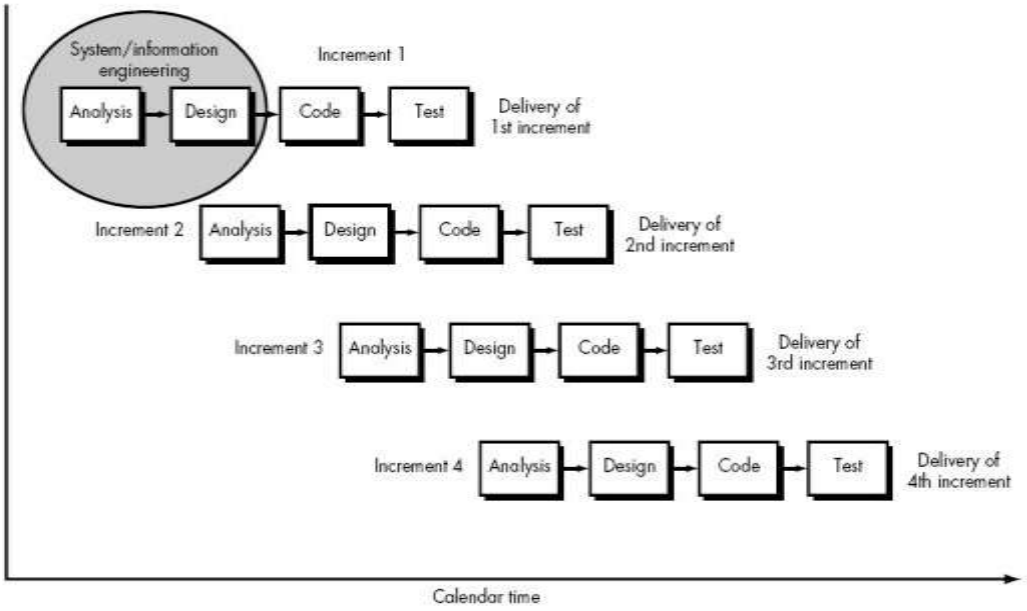


FIGURE
The incremental model

إن نموذج عملية البرمجة ألتزايدي هو تكراري بطبيعته مثل النمذجة الأولية وجميع الأساليب التطورية، ولكنه بخلاف النمذجة الأولية يحرص على تقدم منتج عامل في كل تزايد، تكون التزايدات الأولى "نسخا ممسوخة" عن المنتج النهائي، ولكنها تستطيع تقديم إمكانيات تفيد المستخدم وتقدم أيضا منصة (platform) ليقيمها.

إن التطوير ألتزايدي مفيد، خاصة عند عدم توفر فريق كاف من الموظفين لانجاز الأعمال في الموعد النهائي الذي تم تحديده للمشروع يمكن انجاز التزايدات الأولية بعدد قليل من العاملين، وإذا جرى تقبل النواة المنتج باستحسان، يمكن عندئذ إضافة موظفين جدد (إذا كان ذلك ضروريا) لانجاز التزايد التالي.

بالإضافة إلى ذلك يمكن وضع خطة للتزايدات من اجل إدارة المخاطر التقنية مثلا، قد يتطلب نظام رئيسي توفر عتاد جديد ما يزال قيد التطوير، وتاريخ تسليمه غير مؤكد قد يكون ممكنا

تخطيط التزايدات الأولى بطريقة يمكن فيها تجنب استخدام هذا العتاد، وبالتالي يمكن تقدم وظائف جزئية للمستخدم دون تأخير.

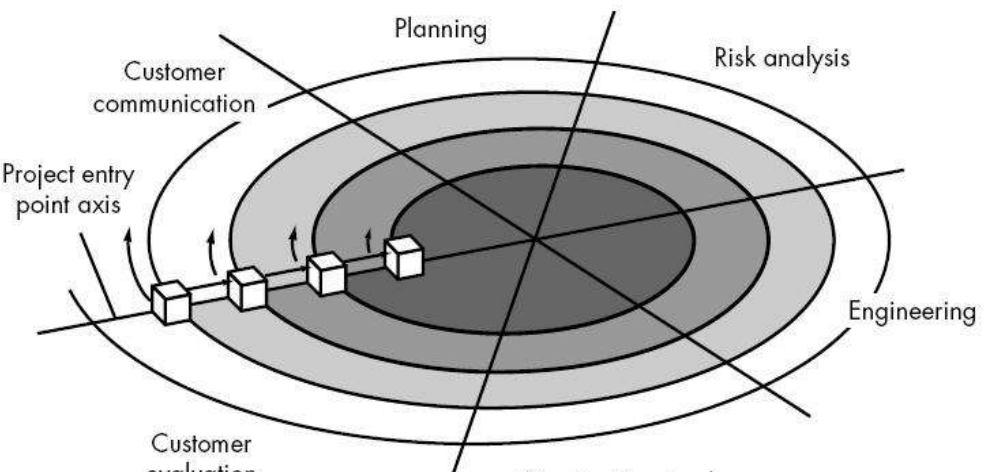
٥-٣ . النموذج الحلزوني:

النموذج الحلزوني (spiral mode) الذي اقترحه أولا Boehm [BOE88] هو نموذج تطوري لعملية البرمجة، ويقرن الطبيعة التكرارية للنمذجة الأولية بالنواحي النظامية والمحكومة للنموذج ألتتابعي الخطي، ويقدم إمكانية تطوير سريع لنسخ تزايدية من البرنامج، يتم تطوير البرنامج حسب النموذج الحلزوني في سلسلة من الإصدارات التزايدية، وقد يكون الإصدار ألتزايدي خلال التكرار الأولي نموذجا ورقياً أو نموذجاً أولياً، ويجري إنتاج نسخ أكثر اكتمالا من النظام الذي تمت هندسته خلال التكرارات التالية.

ينقسم النموذج الحلزوني إلى عدد من نشاطات الهيكل (framework activities) تسمى أيضا منطقة المهمة (task region) [١] ، هناك عادة ما بين ثلاث إلى ست مناطق يمثل الشكل الاحق نموذجا حلزونيا يحتوي على ست مناطق هي :

- الاتصال بالزبون – المهام اللازمة للتواصل الفعال بين المطور والزبون .
- التخطيط – المهام اللازمة لتعريف الموارد، المسارات الزمنية (Timelines)، معلومات أخرى متعلقة بالمشروع.
- تحليل المخاطرة – المهام اللازمة لتقييم المخاطرة التقنية.
- الهندسة – المهام اللازمة لبناء تمثيل أو أكثر للتطبيق.
- التشييد والإصدار (construction & release) – المهام اللازمة لبناء واختبار وتثبيت وتقديم دعم للمستخدم (كالتوثيق والتدريب).
- تقييم الزبون (customer evaluation) – المهام اللازمة للحصول على تعليقات الراجعة بالاعتماد على تقييم تمثيلات البرمجية التي انشات خلال مرحلة الهندسة وتم انجازها خلال مرحلة التثبيت.

FIGURE
A typical spiral model



١
الن

كل منطقة من هذه المناطق الستة مأهولة بسلاسل من مهام العمل التي جرى تكييفها مع خصائص المشروع الذي يجري تنفيذه، يكون عدد مهام العمل الرسمية المتعلقة بها منخفضا في حالة المشاريع الصغيرة، إما في المشاريع الكبيرة ذات الحساسية الأعلى، فتحتوي كل منطقة على مهام عمل أكثر، يجري تعريفها لتحقيق مستوى أعلى من الرسمية، وتطبق في جميع الحالات نشاطات المظلة(مثل إدارة تشكيلة البرمجيات وضمان جودة البرمجيات) .

حالما تبدأ عملية البرمجة التطورية، يتحرك فريق هندسة البرمجيات حول الحلزون باتجاه عقارب الساعة بدءاً من النواة قد ينتج عن الدورة الأولى حول الحلزون تطوير مواصفات المنتج، وقد تستعمل الدورات التالية حول الحلزون لتطوير نموذج أولي، ثم شيئاً فشيئاً تعد نسخ من البرنامج أكثر تطورا ينتج عن كل مرور عبر منطقة التخطيط ضبط لخطة المشروع، ويجري ضبط الكلفة والجدول الزمني بالاعتماد على تعليقات الزبون إضافة إلى ذلك يضبط مدير المشروع عدد التزايدات المخطط لها والمطلوبة لانجاز البرنامج.

خلافا للنماذج التقليدية لعملية البرمجة التي تنتهي عند تسليم البرنامج، يمكن تكييف النموذج الحلزوني لتطبيقه على امتداد كامل حياة البرنامج، يعرف الشكل أعلاه محور نقطة دخول المشروع (point project entry) يمثل كل مكعب يوضع على هذا المحور نقطة البداية لمشروع جديد آخر.

يبدأ مشروع تطوير المفاهيم (concept development project) في نواة الحلزون ويستمر (تحدث تزايدات متعددة على مسار الحلزون الذي يحيط بالمنطقة المظللة المركزية) حتى يكتمل تطوير المفهوم ، تتقدم عملية البرمجة عبر المكعب التالي (نقطة دخول المشروع تطوير منتج

جديد) إذا كان المطلوب تطوير المفهوم ليصبح منتجا حقيقيا وتوضع في بداية لتطوير مشروع جديد، يتطور المنتج الجديد في عدد من التزايدات حول الحلزون متبعا المسار الذي يحيط بالمنطقة التي لها تظليل اخف من تظليل النواة، ويحدث تدفق مشابه لعملية برمجة أنواع أخرى من المشاريع .

في الحقيقة، يبقى الحلزون عند تعريفه بهذه الطريقة يعمل حتى نهاية عمر البرنامج (وضعه خارج الخدمة)، هناك أوقات تكون عملية البرمجة فيها نائمة، ولكن حالما يحدث تغيير ما، تبدأ عملية البرمجة عند نقطة بداية مناسبة (مثل تحسين المنتج).

إن النموذج الحلزوني طريقة واقعية لتطوير أنظمة وبرمجيات واسعة النطاق، ولان البرمجيات تتطور مع تقدم عملية البرمجة، فانه من الأفضل للمطور والزبون فهم المخاطر والقيام بالإجراء المناسب لها في كل مستوى من مستويات التطور، يستخدم النموذج الحلزوني النمذجة الأولية كآلية لتقليل المخاطر، ولكن الأهم من ذلك هو انه يمكن المطور من تطبيق نموذج النمذجة الأولية في أي مرحلة من مراحل تطور المنتج فهو يحافظ على الطريقة التدريجية النظامية التي تقترحها دورة الحياة التقليدية، ولكن يطبقها بإطار تكراري يعكس واقع العالم الحقيقي ويتطلب النموذج الحلزوني اعتباراً مباشراً للمخاطر التقنية في جميع مراحل المشروع، وإذا طبق بالوجه المناسب، وجب أن يقلل المخاطر قبل أن تصبح مشكلة يصعب حلها.

بيد إن النموذج الحلزوني ككل الطرق الأخرى ليس دواءً عاماً، إذ يصعب إقناع الزبون (وخاصة في حالات توقيع عقود) بأنه يمكن التحكم بالطريقة التطورية، ثم إن هذا النموذج يتطلب خبرة جيدة في تقدير المخاطرة، ويعتمد على هذه الخبرة في نجاحه، فإذا لم يكشف وجود مخاطرة رئيسة ولم تعالج، فان المشاكل ستقع دون شك. وأخيراً النموذج بحد ذاته جديد نسبياً ولم يستخدم على نطاق واسع بعد كالنموذج ألتتابعي الخطي أو نموذج النمذجة الأولية، وسيستغرق تحديد فعالية هذا النموذج الجديد الهام بثقة مطلقة عددا من السنوات.

٣-٥ . نموذج التطوير المتزامن

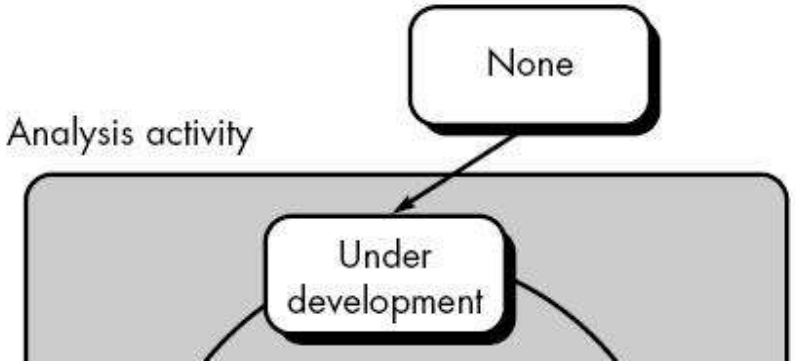
وصف Davis & Sitaram [DAV94] نموذج التطوير المتزامن (concurrent development model)، الذي يسمى أحياناً الهندسة المتزامنة، بالصيغة التالية:

عندما يتابع مدراء المشاريع حالة المشروع من خلال المراحل الرئيسة فقط – من دورة الحياة التقليدية – فإنهم لا يملكون أية فكرة عن أوضاع مشاريعهم هذه الحالة مثال على محاولة متابعة مجموعات معقدة جداً من النشاطات باستخدام نماذج بسيطة جداً. لاحظ مثلاً أنه حتى عندما يكون مشروع (كبير) في مرحلة التشفير، يوجد أشخاص في المشروع مشغولون بنشاطات مترافقة في آن معاً مع عدة مراحل تطويرية أخرى. فمثلاً، يكتب هؤلاء الاشخاص المتطلبات، ويصممون، ويشفرون، ويختبرون، ويجرون اختبارات التكامل (جميعها في آن معاً). وقد بينت نماذج عملية هندسة البرمجيات التي اقترحها Humphrey , Kellner [KEL89, HUM89] التزامن الموجود بين نشاطان تحدث خلال أي مرحلة من المراحل. وتستخدم أحدث أعمال Kellner [KEL91] مخططات حالة تدوين يمثل حالات عملية البرمجة لتمثيل علاقة التزامن الموجودة بين نشاطات مترافقة مع حدث محدد (مثلاً، تغيير المتطلبات في آخر مراحل التطوير)، ولكنها أخفقت في توضيح غنى التزامن الموجود في جميع نشاطات تطوير البرمجيات وإدارتها في مشروع ما. إن معظم نماذج عمليات برمجة تطوير البرمجيات يسيرها الزمن، فكلما تقدم الزمن كان عليك التقدم في عملية التطوير. وبالمقابل يفاد (نموذج عملية البرمجة المتزامنة) باحتياجات المستخدم والقرارات الإدارية ونتائج المراجعة.

يمكن تمثيل نموذج عملية البرمجة المتزامنة تخطيطاً على شكل سلسلة من نشاطات تقنية رئيسية، ومهام وأعمال هندسة برمجيات مرافقة لها. فمثلاُ، يتحقق نشاط الهندسة المعرف للنموذج الحلزوني بتطبيق المهام التالية: النمذجة الأولية و/أو نمذجة التحليل، وتوصيف المتطلبات، والتصميم[١]

يقدم الشكل أدناه تمثيلاً تخطيطياً لنشاط واحد ضمن عملية البرمجة المتزامنة. ويمكن أن يكون النشاط – التحليل – في أي من الحالات[٢] المدونة في أي وقت معطى. وهكذا يمكن تمثيل نشاطات أخرى (كالتصميم أو الاتصال بالزبون) بطريقة مشابهة.تتواجد جميع النشاطات في زمن واحد، ولكنها تكون في حالات مختلفة. فمثلاً ، في وقت مبكر من المشروع ، وأنهى نشاط الاتصال بالزبون (غير مبين في الشكل) دورته (iteration) الأولى ويتواجد في حلة انتظار التغييرات يقوم نشاط التحليل (الذي يوجد في حالة اللاشيء أثناء اكتمال اتصال الزبون) بالانتقال إلى حالة قيد التطوير إلى حالة انتظار التغييرات .

FIGURE
One element of the concurrent process model



يعرف نموذج عملية البرمجة المتزامنة سلسلة من الأحداث التي تسبب الانتقال من حالة إلى أخرى لكل من نشاطات هندسة البرنامج فمثلاً، يجري الكشف عن تضارب في نموذج التحليل خلال المراحل الأولى من التصميم، وهذا يولد حاثة تصحيح نموذج التحليل التي تسبب انتقال نشاط التحليل من حالة الانتهاء إلى حالة انتظار التغييرات .

عادة يستخدم نموذج عملية البرمجة المتزامنة نموذجاً لتطوير تطبيقات العميل/الملقم[1](client/server). ويتألف نظام العميل/الملقم من مجموعة من المكونات الوظائفية. وعند تطبيق نموذج عملية البرمجة المتزامنة على نظام العميل/الملقم، فانه يعرف نشاطات في بعدين (dimensions) [SHE94] **بعد النظام وبعد المكون** وتعالج موضوعات مستوى النظام

[1]) تقسم وظائف البرمجيات في تطبيقات العميل/الملقم بين العميل (عادة كمبيوترات شخصية، PSC) والملقم (كمبيوتر أكثر قوة) يحتوي عادة على قاعدة بيانات مركزية.

باستخدام ثلاثة نشاطات: التصميم والتجميع والاستخدام ويعالج بعد المكون بفعاليتين: التصميم والتحقيق (realization) ويتحقق التزامن بطريقتين:

١. تحدث نشاطات النظام والمكون في آن معاً ، ويمكن نمذجتها باستخدام الأسلوب الحالي التوجه (state-oriented approach) الموصوف سابقاً.
٢. ينجز تطبيق عادي للعميل/ملقم بمكونات عديدة، يمكن تصميم وتحقيق كل منها على نحو متزامن.

إن نموذج عملية البرمجة المتزامنة هو في الحقيقة نموذج قابل للتطبيق في جميع أنواع تطوير البرمجيات، ويقدم صورة دقيقة للحالة الراهنة لمشروع ما. ففي هذا النموذج، تعرف الشبكة من النشاطات بدلاً من تقييد نشاطات هندسة البرمجيات بسلسلة من الأحداث المتوالية. ويتواجد كل نشاط على الشبكة مع نشاطات أخرى في آن معاً . وتسبب أحداث ولدت ضمن نشاط محدد، أو في مكان آخر ما في شبكة النشاطات، انتقالات بين حالات نشاط ما.

٦ . نموذج تجميع المكونات:

تقدم التكنولوجيات الغرضية هيكلا تقنيا لنموذج عملية برمجة يعتمد على المكونات لهندسة البرمجيات، يهتم النموذج ألغرضي التوجه بإنشاء أصناف تغلف كلا من البيانات والخوارزميات المستخدمة لتناول البيانات، وتكون هذه الأصناف الغرضية التوجه قابلة للاستخدام أكثر من مرة في تطبيقات وبنى أنظمة مبنية على كمبيوترات مختلفة، وذلك إذا صممت ونفذت بشكل مناسب.

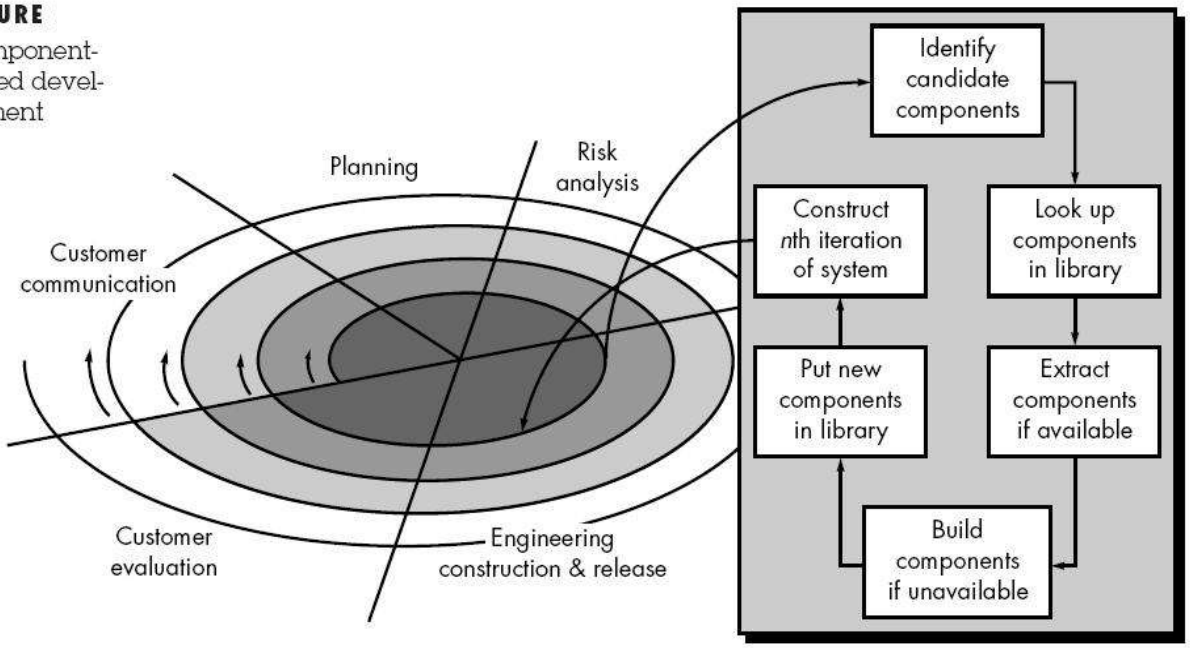
يستخدم نموذج تجميع المكونات (Component assembly model) (الشكل التالي) كثيراً من خصائص النموذج الحلزوني فهو تطوري بطبيعته (NE92) ويتطلب طريقة تكرارية لإنشاء البرمجيات وعلى كل حال، يجمّع نموذج تجميع المكونات التطبيقات من مكونات برمجية جاهزة مسبقاً (prepackaged، تسمى أحياناً اصناف، Classes).

تبدأ الفعالية الهندسية بعملية تمييز الأصناف المرشحة للاستخدام ويتحقق ذلك باختبار البيانات المطلوب أن يعالجها هذا التطبيق، والخوارزميات التي ستستعمل لتحقيق هذه المعالجة[١] . وتوضع الخوارزميات والبيانات الموافقة في صنف ما.

تحفظ الأصناف التي أنشئت في مشاريع هندسة برمجية ماضية في مكتبة أصناف (Class library) أو في مخزن (Repository)، وحالما تميز أصناف مرشحة، يجري البحث في مكتبة

[١]) هذا وصف مبسط لتعريف الصف .

الأصناف ليُعلم: هل هذه الفئات موجودة سابقاً؟ وتستخلص من المكتبة، تجري هندستها باستخدام الطرق الغرضية التوجه. يشكل بعدئذ التزايد الأول من التطبيق المطلوب بناءه باستخدام الأصناف مستخلصة من المكتبة، وأي فئات جديدة مبنية لتحقيق المتطلبات الخاصة بالتطبيق. يعود بعدئذ تدفق عملية البرمجة إلى الحلزون، ويعود أخيراً للدخول إلى تزايدات تجميع المكونات خلال المريرات التالية عبر نشاط الهندسة.



يقود نموذج تجميع المكونات إلى عادة استخدام البرمجيات، وتزود إعادة الاستخدام مهندس البرمجيات بعدد من الفوائد القابلة للقياس، فقد أفادت شركة QSM Associates اعتماداً على دراسات الاستخدام أن تجمع المكونات يقود إلى تقليل زمن دور التطوير بنسبة ٧٠% وتخفيض كلفة المشروع بنسبة ٨٤%، ومؤشر إنتاجية قيمته ٢٦,٢ بالمقارنة بمقياس الصناعة البالغ ١٦,٩ (YOU94). ومع إن هذه النتائج تابعة لقوة ومتانة مكتبة المكونات، فلا يوجد أدنى شك في أن نموذج تجميع المكونات يقدم ميزات هامة لهندسة البرمجيات.

٧. نموذج الطرق الصورية

يشمل نموذج الطرق الصورية (formal methods model) مجموعة من النشاطات التي تقود إلى مواصفات رياضية لبرمجيات الكمبيوتر. وتمكن الطرق الصورية مهندس البرمجيات من وصف نظام معتمد على الكمبيوتر وتطويره والتحقق من صحة عمله بتطبيق تدوين رياضي دقيق. حالياً يطبق بعض مؤسسات تطوير البرمجيات نسخة معدلة عن هذه الطريقة تسمى برمجيات الغرفة النظيفة (Clean Room software engineering) [MIL89,DYE92] .

تقدم الطرق الصورية عند استخدامها خلال التطوير آلية لإزالة العديد من المشاكل التي يصعب حلها باستخدام بقية نماذج هندسة البرمجيات. ويمكن اكتشاف حالات الغموض وعدم الاكتمال وعدم الانسجام (التضارب) وتصحيحها بسهولة أكثر ، وليس باستخدام مراجعة مناسبة وإنما تطبيق التحليل الرياضي عندما نستخدم الطرق الصورية خلال التصميم فإنها تفيد كأساس للتحقق من صحة البرنامج ولهذا فهي تسمح لمهندس البرمجيات باكتشاف وتصحيح الأخطاء التي يمكن أن تبقى غير مكتشفة.

رغم إن نموذج الطرق الصورية لم يصبح بعد طريقة سائدة، إلا انه يعطي برمجيات خالية من العيوب ومع ذلك فهناك مخاوف حول قابلية تطبيقه في بيئة الأعمال منها ما يلي:

١. تطوير النماذج الصورية حالياً مكلفا ويستغرق وقتاً طويلاً.
٢. هناك حاجة ماسة إلى التدريب، لأنه قلة فقط من مطوري البرمجيات قادرين على تطبيق هذه الطرق الصورية.
٣. يصعب استخدام هذه الطرق كآليات للتواصل مع زبون غير متطور تقنياً.

إذا جرى دحض هذه الاتهامات ، فمن الممكن أن يكسب نموذج الطرق الصورية موالين من مطوري البرمجيات، وخاصة المطالبين منهم ببناء برمجيات تهتم كثيراً بالأمان (safety ، كمطوري الكترونيات الطيران للطائرات ومطوري برمجيات المعدات الطبية) والمطورين الذين سيعانون ضائقة مالية إذا حدث خطأ في البرمجيات.