Software Process

محمد الأحمد

Software Engineering is a Layered Technology

The Software Process

مجموعة نشاطات مترابطة مع بعضها قد تتداخل، تؤدي لتطوير نظام برمجي.

الهدف منها تطوير نظام برمجي ضمن وقت مناسب وجودة عالية للوصول لرضا الزبون الذي طلب البرمجية والأشخاص الذين سوف يستخدمونه.

مراحل عملية تطوير البرمجيات:

1) التخصيص (Specification): توصيف النظام والإجرائيات والخدمات التي يقدمها والقيود التي يجب احترامها.

2) التصميم والتنفيذ (Design & Implementation): يشرح كيف يمكننا تحويل الخدمات لنظام فعلي وهو التصميم أما التنفيذ فهو نقل هذا التصميم لكود بلغة معينة.

3) الاختبار (Validation): التحقق من كون البرمجية تنفذ ما يطلبه الزبون.

4) التطوير :(Evolution) تعديل النظام استجابة لحاجة الزبون.

Software Process Model: آلية ورود هذه الأنشطة العامة بطريقة معينة.

Software Process Models

1. **Plane-Driven**: نقوم بعملية التخطيط بشكل كامل ومسبق قبل عملية التنفيذ، عملياً كل الأنظمة القديمة تعتمد هذا النوع من التخطيط، هذا النوع من الإجرائيات يكون فيه الكثير منه **documentation**.
2. **Agile**: التخطيط يتم بشكل تزايدي مع التنفيذ ولا تحوي أي **documentation**.

Plane-Driven Models

بشكل عام معظم الأنظمة الضخمة تُطور باستخدام مزيج من هذه النماذج ولاختيار النموذج المناسب يوجد عدة عوامل:

- طبيعة المشروع أو النظام الذي نريد العمل عليه.
- القيود التي يجب مراعاتها أثناء التطوير.
- فريق العمل المطور وخبراته فيوجد أنواع من **process models** تحتاج خبرات عالية.

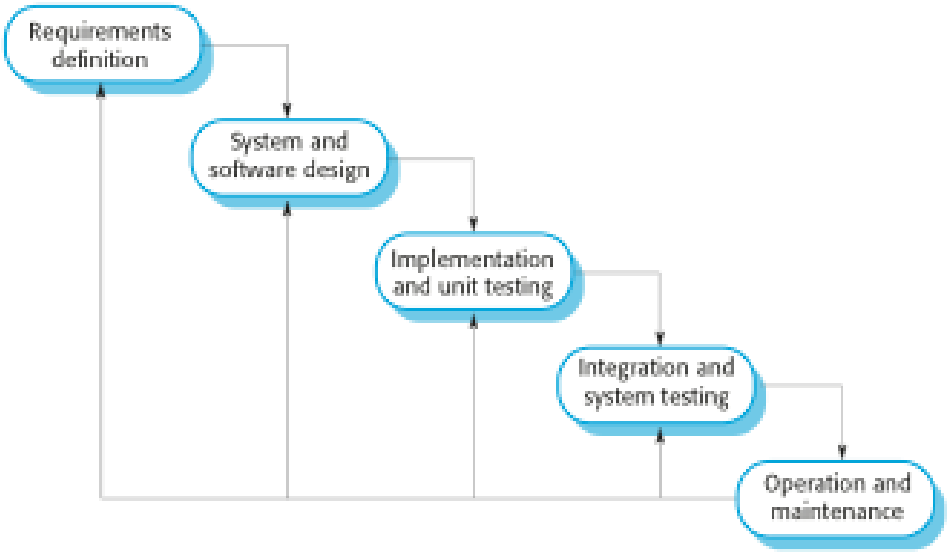
إن إدارة المشروع هي من تختار طريقة التطوير حسب القيود الموجودة والإجرائية الأفضل لتطوير هذا النظام لأن الهدف الأخير هو تطوير نظام بجودة عالية ضمن قيود الكلفة والزمان ولا يمكن القول إن هذه الطريقة أفضل من الأخرى بل حسب ظروف ومتطلبات كل مشروع.

Waterfall Model

العمليات والأنشطة تتم بشكل تسلسلي لا يوجد عودة بالخطوات للوراء ويحوي الأنشطة التالية:

1. تحليل المتطلبات وتحديدها وهي الخدمات والقيود عليها.
2. تصميم النظام.
3. التكويد واختبار الواحدة.
4. تكامل النظام.
5. التشغيل والصيانة.

لماذا سميَّ بالشلالي؟ لأن الأنشطة تأتي بشكل تسلسلي مثل الشلال لا يوجد عودة للخلف كما لا يمكن البدء بأي نشاط من الأنشطة قبل انتهاء النشاط السابق له بشكل كامل وبالتالي يمكن استخدامه عندما تكون المتطلبات معروفة وواضحة وغير قابلة للتعديل.



ميزاته:

- بسيط سهل الفهم لكون العقل البشري يفضل الأفكار التسلسلية.
- مناسب لتدريب الفرق ذات الخبرة البسيطة.
- مناسب للأنظمة ذات الخدمات الثابتة التي لا تتغير مع الوقت.

مساوئه:

- لا يقبل التغيير عند وجود نظام ذو خدمات متغيرة لا يمكن استخدامه.
- أنظمتنا أصبحت معقدة ولم تعد بسيطة فلا يناسبها هذا النموذج.

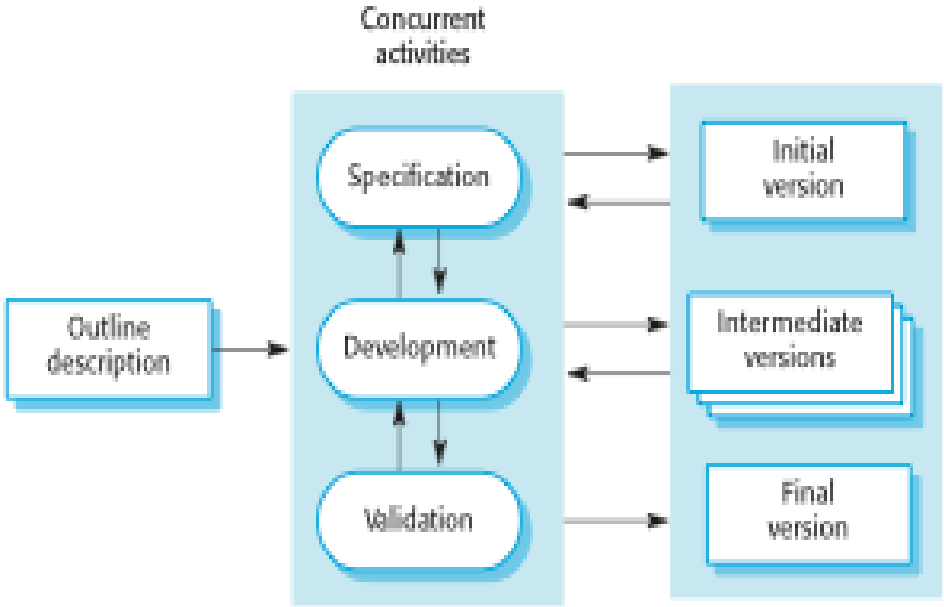
Incremental Development

بعكس الشلالي يشبه القيام ببناء عمارة كل طابق يعتمد على الطابق الذي يسبقه حيث يتم تقديم المنتج عبر سلسلة من الإصدارات لكن في كل عملية تطوير يشمل على النموذج الشلالي كما أننا نعتبر الإصدار السابق ممتاز فلا نقوم بالتعديل عليه وإنما فقط نطوره، نختاره عندما تكون متطلبات النظام متغيرة أمثلة عليه: تطبيقات الويب،Office ...

بالبداية لتطوير نظام بهذا النموذج نقوم بتحديد أولويات الخدمات وفي أول إصدار نقوم بتطوير عشر خدمات "مثلاً" من الخدمات الأساسية لهذا النظام والتي يحددها الزبون وفي كل إصدار جديد نزيد عدد الخدمات وهكذا... ويتم تقديم النظام على شكل مجموعة من الإصدارات.

يتم اختيار النموذج التزايدي عندما نريد الحصول على منتج فعال خلال فترة زمنية قصيرة أما في النموذج الشلالي فيوجد فترة زمنية طويلة بين بدء التطوير والوصول لمنتج كامل.

عملية تحديد المتطلبات والتطوير والتحقق تتم على التوازي أما في المخطط الشلالي تتم على التسلسل وذلك من أجل خدمات قليلة.



ميزاته:

- الحصول على منتج فعال خلال فترة زمنية قصيرة كأسابيع قد لا يحوي كل الخدمات لكنه يحوي الخدمات الأساسية.
- ليس لديه مشكلة مع التغيير في الخدمات والمتطلبات.

مساوئه:

- غير مستقر.
- يوجد كم هائل من التوثيق فكل إصدار يحتاج لتوثيق كامل ومراجعة الإصدار السابق للاطلاع على كل محتواه.

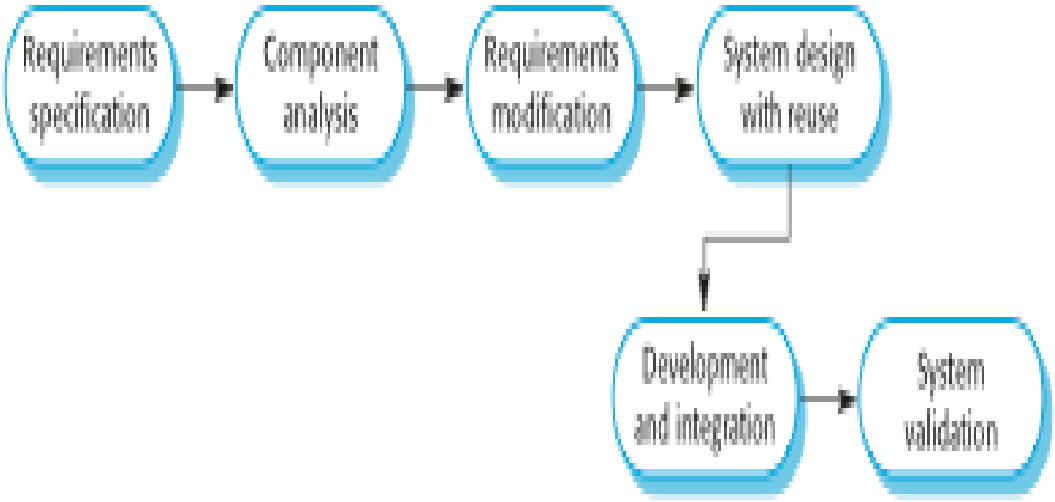
Reuse-Oriented Software Engineering

إعادة الاستخدام مبدأ هام في هندسة البرمجيات ما يميز هذا النموذج أننا لا نقوم بالتطوير من الصفر بعكس النماذج السابقة حيث نبحث عن نظام برمجي موجود في السوق يناسب احتياجات الزبون وقابل لإعادة الاستخدام ونقوم بتعديله ليتناسب مع الخدمات المطلوبة.

تعتمد على Component Analysis أي تحليل وحدة البناء الأساسية والبحث عنها ثم التعديل عليها لتتناسب مع المتطلبات كما يجب تصميم النظام مع فكرة استخدام أجزاء جاهزة وأخيراً عملية التطوير والتكامل.

أثناء التطوير قد نبحث عن جزء ولا نجده عندها نقوم بتطويره يدوياً باستخدام أحد النماذج السابقة.

أثناء تطوير نظام برمجي نحاول الاعتماد على مبدأ إعادة الاستخدام لتوفير الوقت والجهد كما أن الأنظمة المستخدمة تكون ثابتة ومستقرة وخالية من الأخطاء تقريباً لذا فعلياً أي نظام لا يتم تطويره باستخدام نموذج برمجي واحد وإنما مزيج من النماذج.



ميزاته:

- السرعة والكلفة الأقل والجودة عالية نتيجة استقرار هذه الأنظمة.

مساوئه:

عملية التكامل ليست عملية سهلة وهي الجزء الأساسي في هذا النموذج.

Types of software component

من أهم طرق استخدام مبدأ إعادة الاستخدام هي خدمات الويب وهي بعض الخدمات غير الأساسية في الموقع والتي لا يقوم مطورو الموقع بتطويرها وإنما يتم الاشتراك فيها مثل خدمات الطقس وأسعار صرف العملات...ويمكن الحصول على هذه الخدمات عن بعد وتبدو كجزء من النظام ولكنها ليست كذلك.

يوجد مجموعة objects يتم تطويرها ضمن package لتتكامل مع وحدة بيئة العمل مثل .NET , JEE

COTS: مجموعة برمجيات صغيرة موجودة في السوق يمكن شراؤها وتجميعها للحصول على نظام برمجي.

Process Activities

❖ Software Specification:

تعد نهاية مرحلة التحليل التي تشمل الخطوات التالية:

- دراسة الجدوى وتحديد فيما إذا كنا سنقوم بتطوير النظام أم لا وهي جزء من مرحلة التخطيط التي تقوم بها الإدارة مثال: شركة النقل الداخلي ليست بحاجة لنظام حجز إلكتروني "لا يوجد جدوى" أما شركات النقل بين المحافظات يمكن أن تستخدم نظام الحجز الإلكتروني وغالباً ما تكون لدراسة الجدوى عدة أبعاد أهمها:

1. البعد الاقتصادي أي كون عوائد النظام تغطي كلفة تطويره أم لا كما يوجد.
2. بعد آخر هو البعد التقني هل يوجد تقنيات تشغل هذا المشروع وتختلف من بلد لآخر حسب البنية التحتية لكل بلد فمثلاً قد أكون بحاجة لإنترنت متوفر في كل مكان لمشروع ما فلو كان غير متوفر لا يوجد جدوى تقنية.
3. جدوى تشغيلية وجود فريق قادر على تطوير هذا النظام وبعد تطويره هل يوجد فريق عمل لدى الزبون قادر على تشغيل هذا النظام والعمل عليه كل هذه الأسئلة تحتاج لدراسة عند إكمال الدراسة والوصول إلى نتيجة إيجابية نبدأ بتطوير النظام.

- استنباط المتطلبات والخدمات والقيود على أداء هذه المتطلبات والتي يحددها الزبون وقد يكون غير قادر على التعبير عما يطلبه فيجب معرفة ما يريد الزبون "استنباط" ويقوم بذلك محلل النظام.
- تحليل المتطلبات وإزالة التكرار والتناقض: في حال وجود متطلبات مكررة نتخلص منها أما التناقض فيعني وجود تناقض في المتطلبات مثل طلب نظام بنكي يهتم بالأمان وسريع الأداء وهذا لا يمكن تحقيقه لأن الأمان يعني البطء نتيجة كثرة عمليات التحقق فنفضل الأمان على السرعة كونه الأهم في حالة النظام البنكي.
- توصيف المتطلبات: كتابتها ضمن وثائق حيث يقوم محلل النظام بتوصيفه ثم يأتي المصمم ليعمل بناءً على كتابة المحلل ويوجد 3 طرق للتوصيف:

a) الطريقة الطبيعية غير الرسمية وهي الكتابة باللغة العربية أو الإنكليزية وهي الطريقة الأسوأ بسبب إمكانية حدوث سوء فهم.

b) الطريقة الرسمية: وهي طريقة معقدة جداً يستخدم فيها الرياضيات بشكل كامل وتستخدم لتوصيف الأنظمة الحرجة التي لا تقبل الخطأ مثل أنظمة الطيران كونها تقبل تفسير واحد فقط.

c) طريقة النصف رسمية والتي تستخدم النماذج "المخططات" مثل مخططات UML، وسنستخدم Use Case Diagram لتوصيف المتطلبات الوظيفية وإظهارها بطريقة رسومية بدل الكتابة.

- التحقق من الكتابة: التحقق من كون المتطلبات موافقة لما يريده الزبون كما يوجد طرق تحقق أخرى من قائمة المتطلبات مثل توصيف المتطلبات بشكل كامل وعدم وجود تناقض والكتابة بشكل واضح.

خرج مرحلة التحليل وثائق ومخططات غالباً تسمى SRS = Software Requirement Specification توصيف المتطلبات البرمجية تحوي هذه الوثائق أيضاً models مثل Use Case Diagram وActivity Diagram.

SRS: هي وثيقة تلخص مرحلة التحليل يكون فيها جميع المتطلبات الوظيفية أي الخدمات المطلوب تقديمها ضمن النظام والقيود التي سيطور ضمنها هذا النظام وتكون خرج لمرحلة التحليل ودخل لمرحلة التصميم.

❖ Design & Implementation:

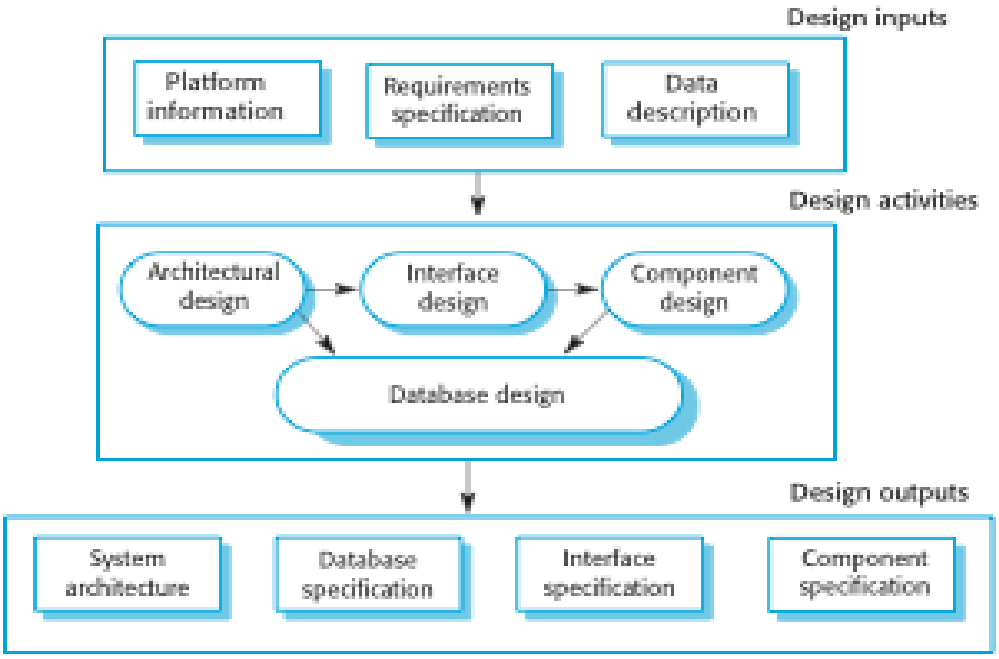
يركز على كيفية تطوير النظام لتحقيق الخدمات المطلوبة منه وهي أهم مرحلة من مراحل تطوير النظم البرمجية وهي المرحلة الهندسية الوحيدة حيث أن مرحلة التحليل ذات منحى اجتماعي أما الإبداع فيتركز في مرحلة التصميم لرسم المخططات الهندسية وحل المشاكل، ماذا يحدث في التصميم؟ يتم وضع خطة للنظام وكيف يتوزع والخوارزميات المستخدمة ثم تأتي مرحلة التكويد لتحويل هذه الخطة إلى كود برمجي بلغة برمجية.

يقسم التصميم إلى مرحلتين أساسيتين:

I. التصميم المعماري: البنية العامة للنظام والأنظمة الجزئية التي يتكون منها وكيف تتواصل هذه الأنظمة الجزئية فيما بينها حيث يختار التصميم المناسب للنظام من ضمن عدة تصاميم جاهزة بالإضافة للمساته الإبداعية لكن بشكل عام جميع التصاميم جاهزة ويهتم بتصميم قاعدة المعطيات والواجهات والاهتمام بشكلها وجماليتها.

II. التصميم التفصيلي: وضع الخوارزميات مناسبة لعمل كل نظام جزئي.

ملاحظة: تصميم الواجهات يشمل تصميم الواجهات الخارجية والواجهات الداخلية التي تحدد طرق تفاعل الأنظمة الداخلية مع بعضها.

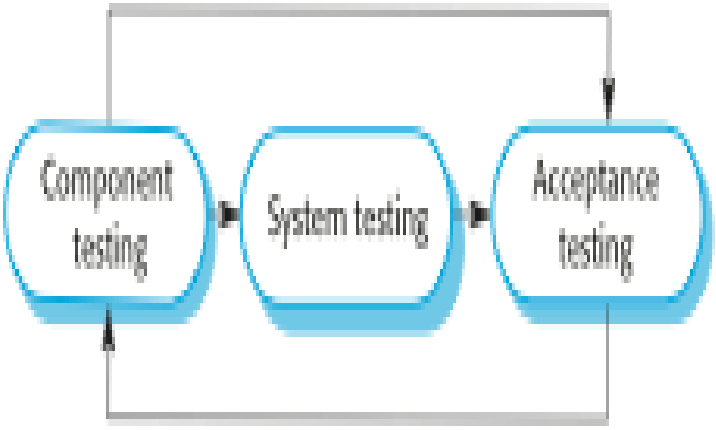


خلال مرحلة التكويد توجد مرحلة Unit Testing "اختبار الوحدة" وهي مهمة المبرمج بالإضافة لتحويل التصميم إلى كود وهي تعني اختبار الكود بعد الانتهاء من كتابته عن طريق إدخال بيانات بهدف إصلاح الأخطاء في حال وجودها.

❖ Validation:

يبدأ باختبار الواحدة لإصلاح الأخطاء في الأجزاء البسيطة والمسؤول عنه المبرمج ثم لدينا اختبار التكامل أي عمل الأجزاء مع بعضها بعدها ننتقل لاختبار النظام بشكل كامل، المبرمج يساعد التكامل فقط مساعدة حيث يوجد متخصصون لإجراء اختبار التكامل أما في اختبار الواحدة فالمبرمج هو المسؤول الوحيد.

الهدف من عملية الاختبار هو اكتشاف الاخطاء قدر الإمكان لأنه لا يمكن البرهان على عدم وجود أخطاء بل دائماً يوجد أخطاء وبالتالي نعتبر التحقق ناجح بقدر اكتشاف الأخطاء لنصل إلى تسليم الزبون نظام خالٍ من الأخطاء قدر الإمكان للوصول إلى منتج يحصل على رضا الزبون.



اختبار القبول (Acceptance Testing) هو اختبار يقوم به الزبون للتأكد من مطابقة النظام للمتطلبات الموضوعة مسبقاً وله نوعان: ألفا يختبر النظام في الشركة المطورة وبيتا يختبر النظام في بيئة العمل.

❖ Software Maintenance:

مرحلة التشغيل والصيانة أو مرحلة التطوير يتم فيها إضافات أو إصلاحات وهي أطول فترة زمنية وأكثرها كلفة .

التعامل مع التغيير

نحن نعيش في محيط قابل للتطوير بشكل لحظي وخاصة المتطلبات ودائماً الزبون رأيه متقلب وطلباته كثيرة وكما رأينا سابقاً أن النموذج الشلالي غير قادر على التعامل مع التغيير لذلك يجب استخدام نماذج أخرى مثل النموذج التزايدي أو أحد النماذج التالية:

Prototype

وهو ما يسمى النمذجة البدائية أو الأولية يستخدم عند وجود نظام معقد للبدء بشكل بسيط ثم زيادة التعقيد ويتم التخلص منه عند الوصول للمتطلبات وله نوعان:

1. Throw-away: تقنية من تقنيات جمع المتطلبات والهدف منها فقط جمع المتطلبات ونستخدم هذه الطريقة عندما يكون الزبون غير قادر على تحديد متطلبات النظام عندها نحاول رسم رسومات بسيطة أحياناً تمثل واجهات للنظام.
2. Exploratory Development: نبدأ النظام بنموذج بسيط ثم يأخذ بالتعقيد.

الفرق بين النموذج التطوري والنموذج التزايدي: في النموذج التزايدي نبدأ بالأكثر أولوية بالنسبة للزبون بينما في النموذج التطوري نبدأ بالخدمات الأبسط ثم نزيد تعقيد الخدمات.

مساوئه:

- ضياع الوقت بسبب التعديلات على النماذج دون الحصول على رضا الزبون.

ميزاته:

- سهولة الاستخدام لكون الزبون مشرف على جميع مراحل التصميم والتعديل يتم وفقاً لآراء الزبون.
- الحصول على نظام مطابق لحاجات الزبون.
- تحسن نوع التصميم.

- سهولة الصيانة.
- تقليل جهود التطوير.

Incremental delivery

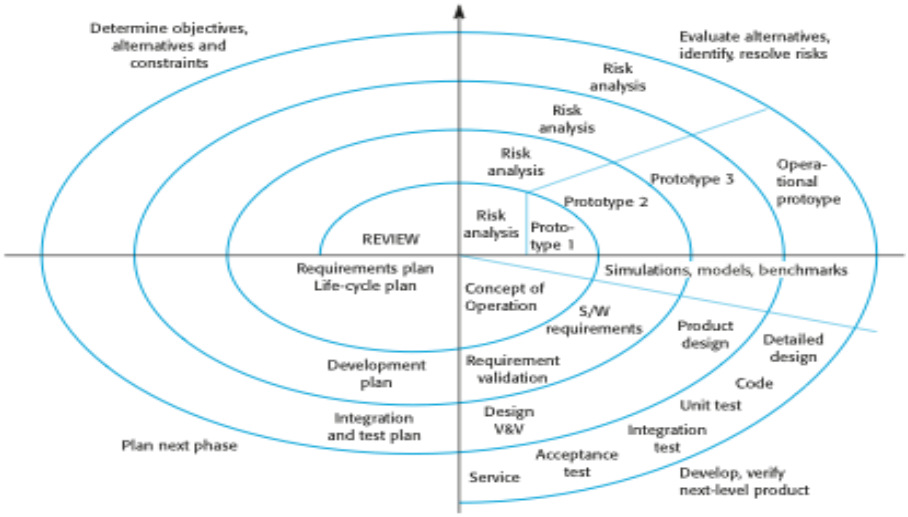
توصيل الخدمات بشكل تزايدي نبدأ بالخدمات الأساسية ثم تأخذ الخدمات بالتزايد كما يمكن أن نقوم بتجميد الخدمات التي لا نرغب فيها وليس لديه أي مشكلة في التغيير.

ميزاته:

- كل إصدار يجد الزبون ميزات جديدة.
- الإصدارات الأولية تعمل دون مشاكل وبوقت قصير.
- أقل خطورة في حال فشل النظام.
- إعطاء أهمية أكبر للخدمات المهمة للنظام من حيث التحقق.

Boehm's spiral model

النموذج الحلزوني وهو نموذج معقد وضخم يستخدم في الأنظمة الحديثة بالمطلق والكبيرة الضخمة ليس للأنظمة البسيطة ويحتاج فريق عمل ذو خبرة عالية وهو قليل الاستخدام.



وهو عبارة عن مجموعة من التكرارات نبدأ بالاتصال مع الزبون لمعرفة متطلباته ثم تحليل المخاطر وهو النموذج الوحيد الذي يحوي هذه الخطوة بعدها تخطيط المشروع وإنجازه ثم اختباره وننتهي بالتحقق من قبل الزبون.

يتألف من عدة لفات وليس شرط عند الانتهاء من لفة الحصول على منتج جاهز للاستخدام وإنما قد نحتاج أكثر من لفة، الفرق بينه وبين النموذج التزايدي أنه في التزايدي كلما انتهيت من تزايد نحصل على نظام جاهز للاستخدام أما بالنموذج الحلزوني ليس شرط عند الانتهاء من لفة الحصول على نظام جاهز.

ملاحظة: النظام الحلزوني هو النظام الوحيد الذي يتضمن مرحلة تحديد مخاطر.

ميزاته:

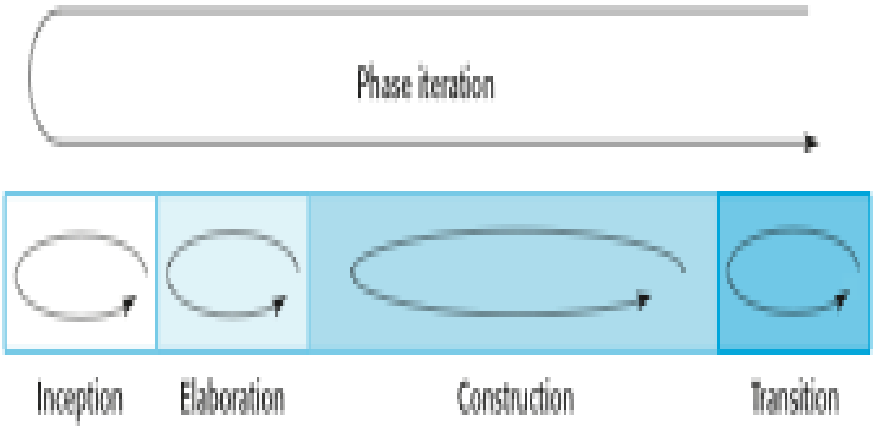
- ليس لديه مشكلة مع التغيير.

مساوئه:

- يحتاج إلى خبرة عالية.
- غير نافع في الأنظمة البسيطة.

The Unified Process

أحدث نموذج إجرائي ظهر بعد ظهور البرمجة غرضية التوجه لدعمها وهو نموذج مهم للحصول على نظم ذات جودة عالية بكلفة ووقت مقبولة لكنه يحتاج لخبراء ويعمل على التوازي أهم ما يميزه أنه يعمل على نوعين من التكرارات: تكرارات مرحلية وتكرارات كلية ويشمل المراحل التالية: عملية الاستهلال، عملية الإعداد، عملية البناء وعملية الانتقال (التحويل).



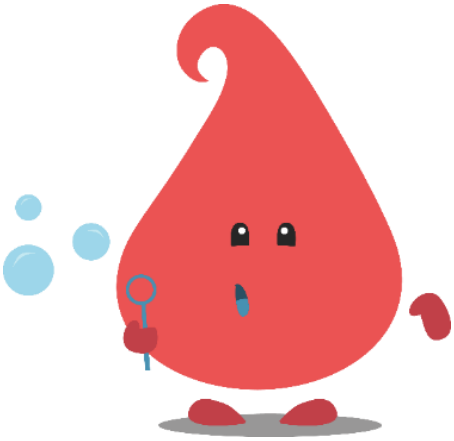
يتم في مرحلة الاستهلال التحقق من خدمات النظام فيثبت الخدمات الضرورية والأساسية ويحدد حدود النظام من حيث الخدمات ثم تليه مرحلة الإعداد ثم البناء وهي أطول مرحلة ويتم فيها التحليل والتصميم والتكويد والاختبار وأخيراً مرحلة النقل وهي المرحلة التي يتم فيها تثبيت النظام في بيئة التشغيل لدى الزبون.

بماذا يختلف عن النماذج الأخرى؟ جميع المراحل تتم على التوازي حيث نبدأ من حدود النظام وهي الخدمات البسيطة والأساسية ثم نمرر هذه الأجزاء البسيطة للمرحلة التالية وهكذا فبعد فترة زمنية معينة جميع المراحل تعمل على التوازي ويكون لدينا سرعة في عملية الإنجاز.

الجودة العالية تأتي من وجود نوعين من التكرارات حيث يوجد إصلاح دائم للأخطاء بعكس التزايدي لأننا في التزايدي نبني على ما سبق أما في التكراري نعيد تكرار المراحل من البداية ونوسع فيه لذا عملياً الأنظمة التي تُطور بهذا النموذج هي أنظمة ذات جودة عالية نتيجة المراجعة التي تتم بشكل مكرر.

عوامل اختيار النموذج المناسب:

1) المتطلبات: هل هي ثابتة أم متغيرة.
2) فريق التطوير: هل لديه خبرات كافية.
3) المستخدمون.
4) نوع المشروع والأخطار الموجودة.



انتهت المحاضرة..