هندسة برمجيات ١

المحاضرة الثانية

د. اناس ليلى

الفصل الدراسي الاول ٢٠٢٥ / ٢٠٢٦

جَامعة المَنارة
MANARA UNIVERSITY

مفردات المقرر

- مقدمة في هندسة البرمجيات
- الإجرائيات البرمجية Software Processes
- التطوير البرمجي الرشيق Agile Software Development
- هندسة المتطلبات Requirements Engineering
- نمذجة النظام System Modeling
- التصميم والتنجيز Design and Implementation

هندسة البرمجيات هي فرع من **فروع الهندسة يقوم على مجموعة أسس وقواعد** تهدف الى تطوير برمجيات عالية الجودة تلبي احتياجات المستخدمين ومتطلباتهم على جميع المستويات. تهتم هندسة البرمجيات بتكوين البرنامج في كل مراحله من التحليل وحتى الصيانة.

كما تعرف بانها تطبيق مبادئ الهندسة والطرق العلمية المنهجة بهدف إنتاج برمجيات موثوقة، فعّالة، قابلة للصيانة، قابلة للتطوير تُسلّم ضمن الوقت والتكلفة المحددين.

في هندسة البرمجيات، بناء النظام البرمجي ليس مجرد كتابة شفرة، وإنما هي عملية إنتاجية لها عدة مراحل أساسية وضرورية للحصول على المنتج بأقل كلفة ممكنة وأفضل أداء محتمل. يطلق على هذه المراحل اسم دورة حياة تطوير البرمجية (Software development Lifecycle SDLC) وهناك الكثير من التصورات والنماذج في هندسة البرمجيات تصف عملية إنتاج برنامج والخطوات اللازمة لذلك

Software Project Stakeholder

في هندسة البرمجيات، يُعرَّف صاحب المصلحة بأنه أي فرد أو مجموعة أو منظمة لها مصلحة راسخة في مشروع برمجي، سواءً أكانوا مشاركين أو متأثرين بنتائجه بشكل مباشر أو غير مباشر.

يشمل ذلك الأشخاص الذين قد يؤثرون أو يتأثرون بنجاح المشروع أو فشله.

يلعب أصحاب المصلحة دورًا حاسمًا طوال دورة تطوير البرمجيات، من البداية إلى النهاية، من خلال تقديم الرؤى والملاحظات والموارد التي تُشكل مسار المشروع ونتائجه.

يمكن تصنيف أصحاب المصلحة إلى مجموعتين رئيسيتين وثانويتين. لأصحاب المصلحة الرئيسيين تأثير مباشر على المشروع، وغالبًا ما يكونون مفتاح نجاحه؛ وهم يشملون العملاء الذين يحددون متطلبات المشروع ونطاقه، ومديري المشاريع الذين يشرفون على العملية، ومحللي الأعمال الذين يحددون احتياجات المشروع، والمطورين ومهندسي ضمان الجودة المسؤولين عن بناء البرنامج والتحقق من صحته، ومصممي واجهة المستخدم/تجربة المستخدم الذين يضمنون سهولة استخدام المنتج. يشارك هؤلاء الأفراد بفعالية في عملية التطوير ويساهمون بشكل مباشر في إنشاء المنتج وجودته. أما أصحاب المصلحة الثانويون، فلديهم علاقة غير مباشرة بالمشروع، ولكن لا يزال بإمكانهم التأثير على القرارات أو سمعة المنتج. تشمل هذه المجموعة المستخدمين النهائيين الذين قد يقدمون ملاحظاتهم أثناء الاختبار.

الاجرائية البرمجية software process تركز على "كيف" يُبنى البرنامج وهي مجموعة من الأنشطة التي تقود لإنتاج المنتج البرمجي. (تُقسّم الجهد الكلي إلى خطوات أصغر أو عمليات فرعية تهدف إلى ضمان نتائج عالية الجودة).

نموذج الإجرائية البرمجية (Software Process Model) عبارة تمثيل مبسط للإجرائية البرمجية، يُظهر الأنشطة المتضمنة فيها وتسلسلها. وهناك العديد من نماذج الاجرائيات المعرفة بشكل جيد، ولا تزال الأبحاث الحديثة جارية لإيجاد المزيد من النماذج الأفضل.

Computer-aided software engineering أو (CASE)

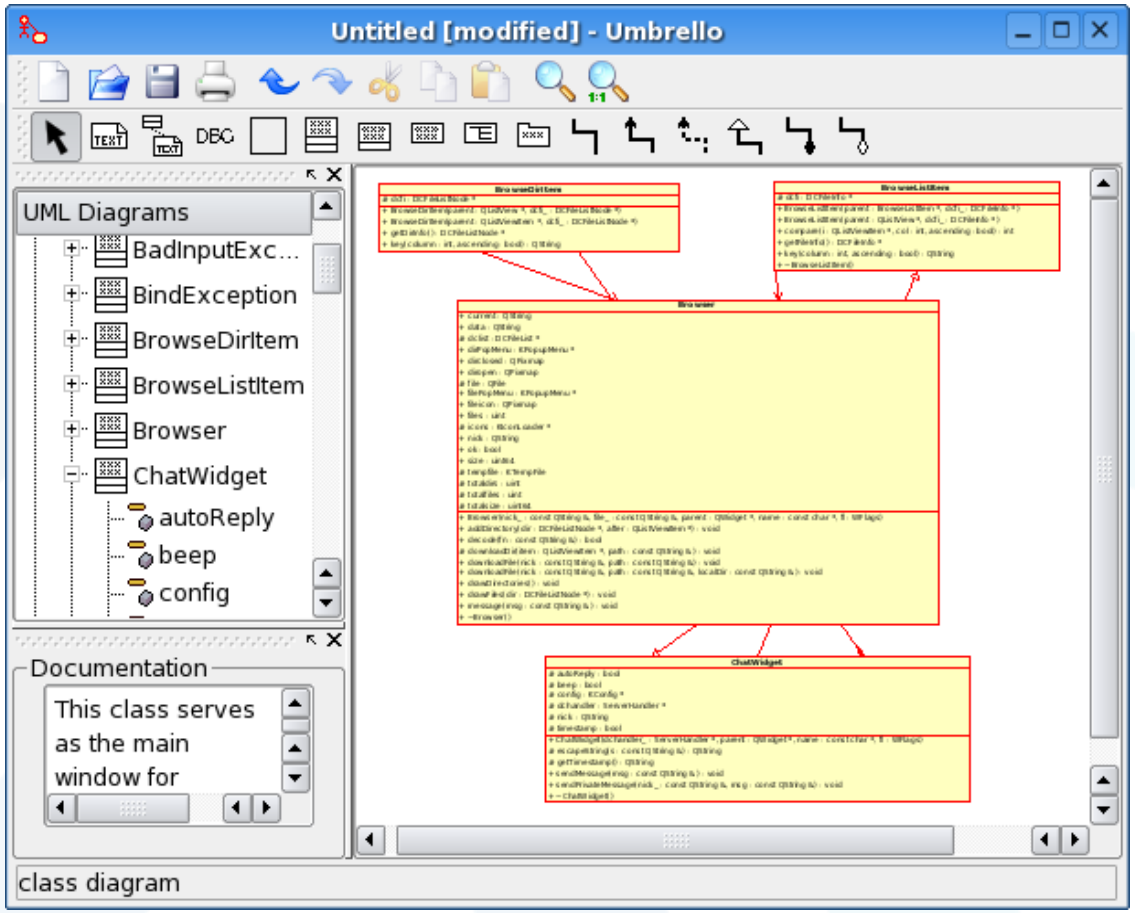
هندسة البرمجيات بمساعدة الحاسوب: يشير المصطلح إلى استخدام أدوات برمجية ضمن عملية تصميم وتطوير البرمجيات.

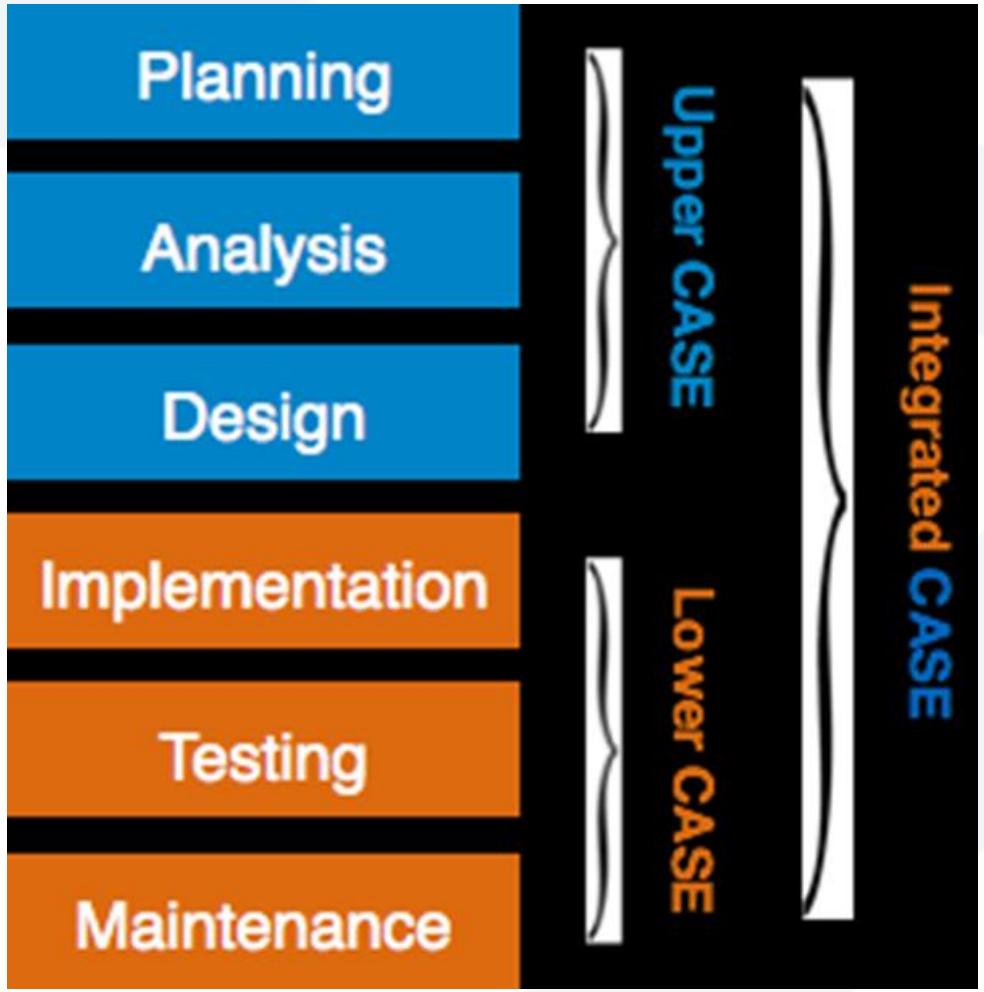
Computer-aided software engineering (**CASE**) is a domain of software tools used to design and implement applications.

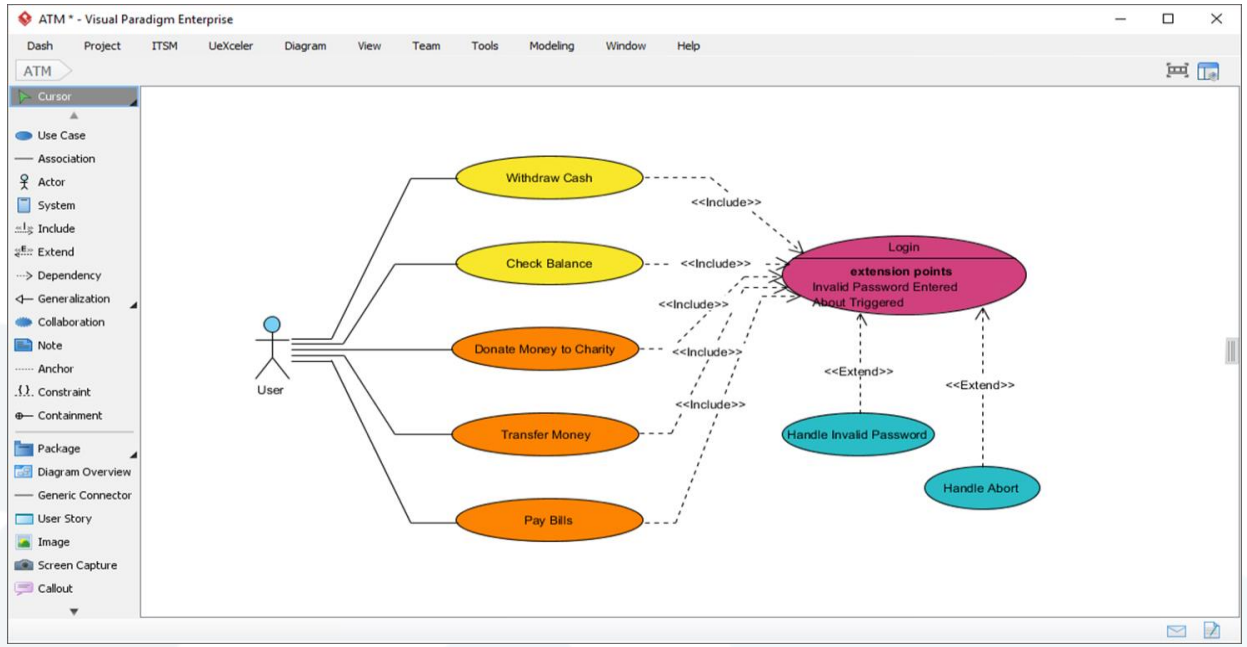
CASE tools support specific tasks in the software development life-cycle.

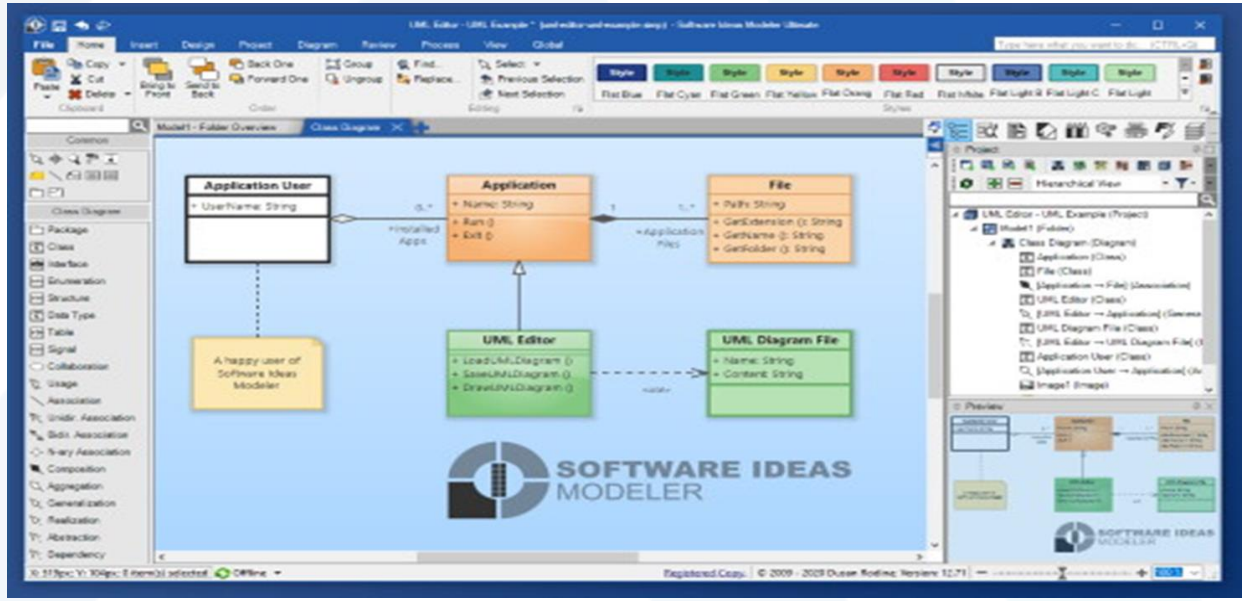
common way to distinguish CASE tools is the distinction between Upper CASE and Lower CASE. Upper CASE tools support the early stages of development, including analysis and design, and are typically used by business analysts and system.

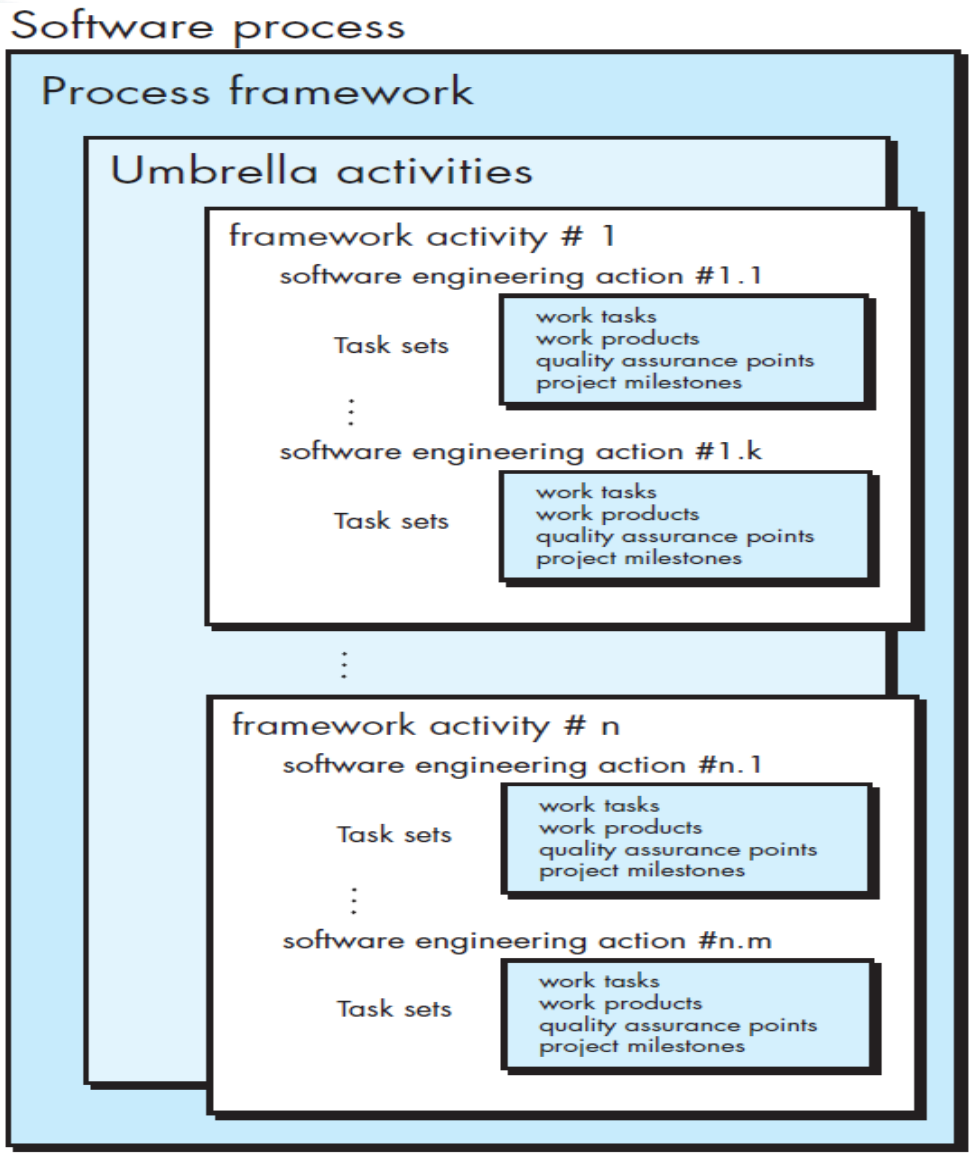
Lower CASE tools focus on the later stages of the life cycle—implementation, testing, and maintenance




Planning
Analysis
Design
Implementation
Testing
Maintenance
Upper CASE
Lower CASE
Integrated CASE


ATM * - Visual Paradigm Enterprise
Dash
Project
ITSM
UeXceler
Diagram
View
Team
Tools
Modeling
Window
Help
ATM
Cursor
Use Case
Association
Actor
System
Include
Extend
Dependency
Generalization
Collaboration
Note
Anchor
Constraint
Containment
Package
Diagram Overview
Generic Connector
User Story
Image
Screen Capture
Callout
Withdraw Cash
Check Balance
Donate Money to Charity
Transfer Money
Pay Bills
User
<<Include>>
Login
extension points
Invalid Password Entered
About Triggered
<<Extend>>
Handle Invalid Password
Handle Abort


Application User
+ UserName: String
Application
+ Name: String
+ Run ()
+ Exit ()
File
+ Path: String
+ GetExtension (): String
+ GetName (): String
+ GetFolder (): String
UML Editor
+ LoadUMLDiagram ()
+ SaveUMLDiagram ()
+ DrawUMLDiagram ()
UML Diagram File
+ Name: String
+ Content: String
A happy user of Software Ideas Modeler
SOFTWARE IDEAS MODELER



يوصف الشكل الموضح جانباً إطار عمل الإجرائية البرمجية Software Process Framework

يوفر إطار عمل الاجرائية البرمجية نهجًا منظمًا لتطوير البرمجيات، ويشكل أساسًا لعملية هندسة البرمجيات بأكملها.

ينشئ هذا الإطار مجموعة صغيرة من أنشطة الإطار القابلة للتطبيق على جميع مشاريع البرمجيات، بغض النظر عن حجمها أو تعقيدها، ويتضمن أنشطة شاملة تغطي دورة حياة التطوير بأكملها.

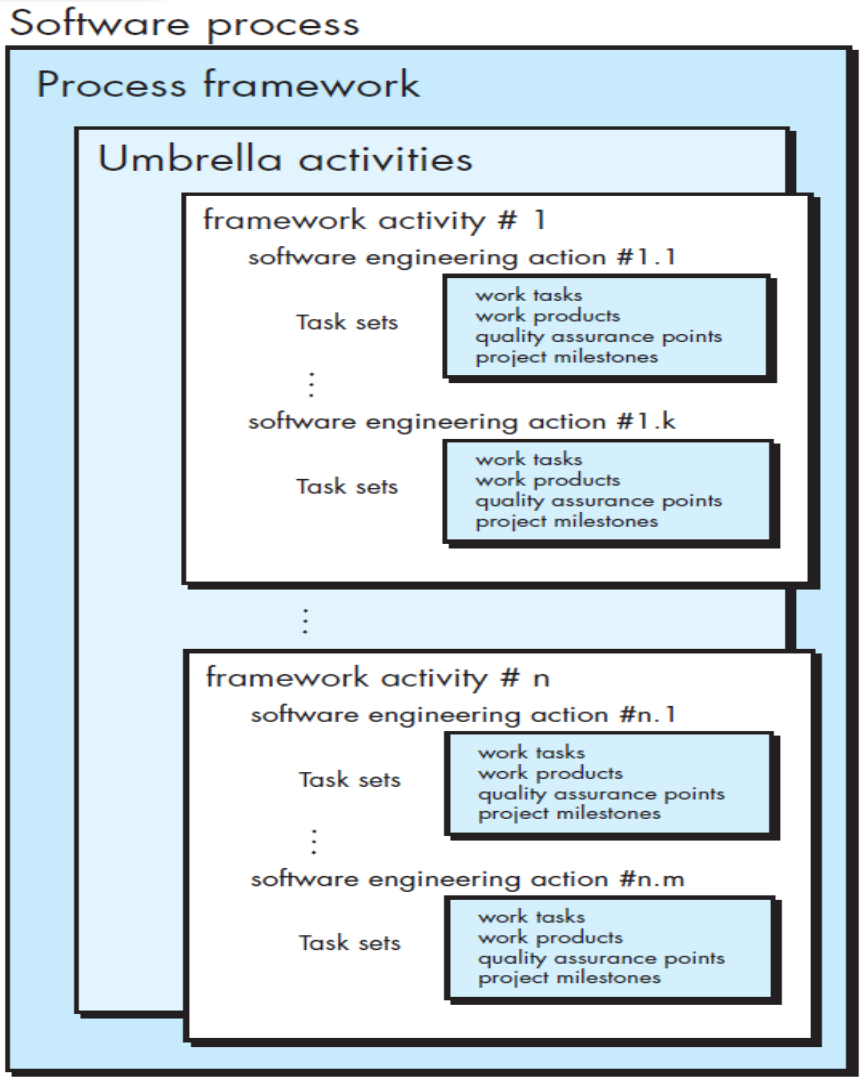
أنشطة الإطار الخمسة العامة هي: التواصل، والتخطيط، والنمذجة، والبناء، والنشر.

يتكون كل نشاط من أنشطة الإطار من مجموعة من الإجراءات، والتي تتكون بدورها من مجموعات مهام، ومنتجات العمل، ونقاط ضمان الجودة. يتضمن التواصل التفاعل مع العملاء وأصحاب المصلحة لجمع المتطلبات وتوضيحها .يركز التخطيط على وضع خطة عمل، وتحديد المخاطر الفنية، وحصر متطلبات الموارد، وتحديد الجداول الزمنية. تشمل النمذجة إنشاء نماذج معمارية وتصميمية لفهم المشكلة بشكل أفضل وتوجيه الحل.

يتضمن البناء توليد الشيفرة البرمجية واختبارها لبناء البرنامج والتحقق من صحته.

يشير النشر إلى تسليم البرنامج للعملاء للتقييم والحصول على الملاحظات، مما قد يؤدي إلى تحسينات متكررة.

كما يتضمن الاطار أنشطة المظلة



Source: (Pressman and Maxim, 2015)

تمثل نشاطات المظلة (Umbrella activities): مجموعة من الأنشطة التي يتم تطبيقها خلال مراحل تطوير البرمجيات، وهي تتعلق بـ:

- إدارة المخاطر (Risk management)
- إدارة جودة البرمجيات (Quality management)
- المراجعات الفنية الرسمية (Formal Technical reviews)
- وغيرها

إدارة المخاطر: عبارة عن سلسلة من الخطوات لمساعدة فرق تطوير البرمجيات على فهم وإدارة حالة عدم اليقين وتحديدها، وتقييم احتمالية حدوثها، وتقييم تأثيرها، ووضع خطة طوارئ "في حال حدوث المشكلة".

ضمان جودة البرمجيات: تُحدد وتُنفذ الأنشطة اللازمة لضمان جودة البرمجيات

المراجعات الفنية الرسمية: تُقيّم هذه المراجعة نواتج عمل هندسة البرمجيات في محاولة للكشف عن الأخطاء ومعالجتها قبل انتقالها إلى النشاط التالي.


Risk
Managemet
Software
Project Tracking
& Control
Measurment
& Metrics
Formal
Techinical
Reviews
Umbrella
Activities
Re-Usability
Management
Software
Quality
Assurance
Document
Prepration
& Production
Software
Configuration
Managements

يقسم النماذج الإجرائية لتطوير البرمجيات إلى صنفين أساسيين:

- نماذج الإجرائيات التي تعتمد على التخطيط (Plan-driven Processes)

حيث يتم التخطيط لجميع أنشطة الإجرائية مسبقاً ويتم قياس التقدم مقابل هذه الخطة وتُعدّ مناسبة للمشاريع التي تتطلب ثباتًا في المتطلبات وتحتاج إلى تخطيط دقيق.

تعدّ هذه النماذج مناسبة للمشاريع التي لا تتطلب تغييرات متكررة في المتطلبات، حيث يُركز الهدف على إدارة الإطار الزمني وميزانية المشروع بدلاً من التسليم السريع للمنتج. تُعدّ من أبرز عيوبها قلة المرونة في مواجهة التغييرات، حيث يصعب إجراء تعديلات بعد اكتمال مرحلة معينة، مما يجعلها أقل ملاءمة للمشاريع ذات المتطلبات غير الواضحة أو المتغيرة.

Plan-driven process is **a process where all the activities are planned first, and the progress is measured against the plan**.

• الإجرائيات الرشيقة (Agile Processes)

○المنهجيات الرشيقة في هندسة البرمجيات هي **نهج تطوير دوري يعتمد على التكرارات القصيرة بدلاً من عملية تسلسلية واحدة**، تركز هذه المنهجيات على التعاون بين الفرق، والعمل الجماعي، والاستجابة السريعة للتغييرات، مع التركيز على تقديم قيمة مضافة للعميل بشكل مستمر. تُعد المنهجية الرشيقة طريقة تطوير دورية بدلاً من إصدار منتج شامل واحد في النهاية وتعتبر المنهجية الرشيقة أكثر مرونة مقارنة بالنماذج التي تعتمد على الخطة، حيث يُمكن للعميل رؤية المنتج بعد كل تكرار

○تتمحور هذه الإجراءات حول بناء المنتج على مراحل قصيرة، تُعرف بـ "الدورات" (Sprints)، والتي تستمر عادةً بين أسبوعين إلى أربعة أسابيع، بهدف تسليم منتجات أولية صالحة للاستخدام بشكل دوري. يُعدّ التسليم المبكر والمستمر أحد أولويات هذه المنهجية، حيث يُقدّم كل مرحلة منتجًا حقيقيًا يمكن للعميل التفاعل معه وتقديم ملاحظاته. هذا يُمكّن الفريق من التفاعل مع العميل بشكل مستمر، وتعديل المخرجات بناءً على ردود الفعل، مما يقلل من المخاطر ويزيد من رضا العميل.

○البساطة، هي جزء أساسي وحيوي ومهم في الـ Agile، أي تقليص الأعمال الغير مهمة والغير ضرورية.

○تعتمد الأجايل على الاجتماعات القصيرة لفريق العمل حيث تُنَاقَشُ فيها المستجدات والمشاكل البرمجية التي تمت مواجهتها أثناء العمل. يقوم كل من أعضاء الفريق بعد ذلك بالتعاون لحل هذه المشاكل. كما تعمد عملية بناء البرمجيات إلى تكريس معظم الوقت لإنشاء البرامج بدلاً من إعداد التقارير الخاصة بتوثيق هذه البرامج.

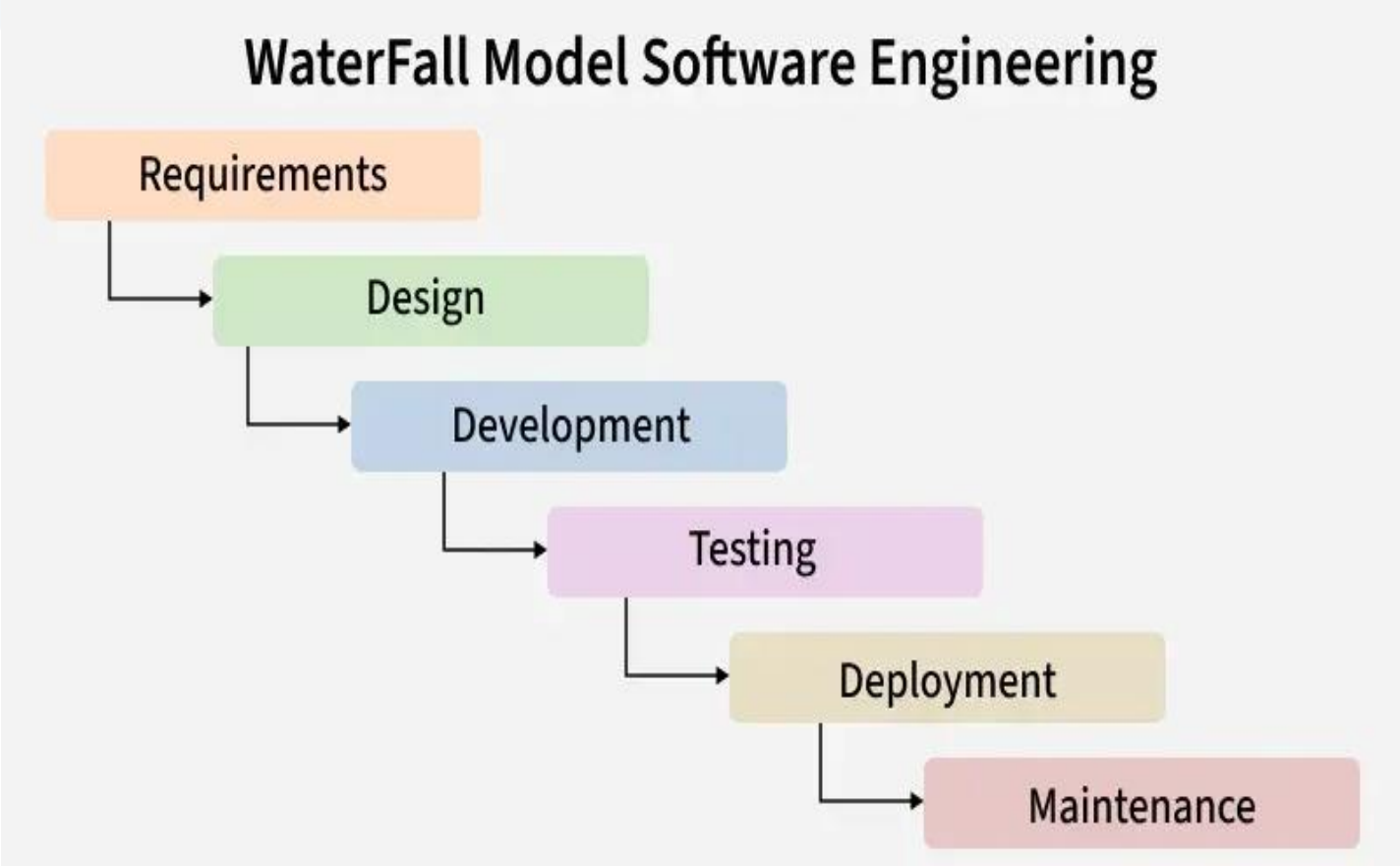
○تهدف هذه المنهجية الى إرضاء العميل عن طريق التسليم المبكر والمتواصل لبرمجيات ذات قيمة والترحيب بتغيير المتطلبات.

Agile Development Model is a combination of iterative and incremental models, that is, it is made up of iterative and incremental models.

نماذج الإجرائيات المعتمدة على التخطيط Plan-driven Processes Models

- نموذج الشلال Waterfall Model
- النموذج التزايدي Incremental Model
- النموذج الحلزوني Spiral Model
- النموذج الأولي Prototyping Model

Waterfall Model



نموذج الشلال هو نموذج متسلسل يقسم تطوير البرمجيات إلى مراحل محددة مسبقًا. يجب إكمال كل مرحلة قبل أن تبدأ المرحلة التالية دون أي تداخل بين المراحل. تم تصميم كل مرحلة لأداء نشاط محدد خلال مرحلة SDLC . تم تقديمه في عام ١٩٧٠ من قبل ونستون رويس.

يُعد مناسبًا للمشاريع ذات المتطلبات الثابتة والواضحة.

اشتُقت تسمية النموذج من التدرج الثابت الذي تتميز به مراحله من الأعلى إلى الأسفل، مشابهًا لتدفق الماء في الشلال، حيث لا يمكن الانتقال إلى المرحلة التالية إلا بعد اكتمال المرحلة السابقة تمامًا

نموذج الشلال Waterfall Model

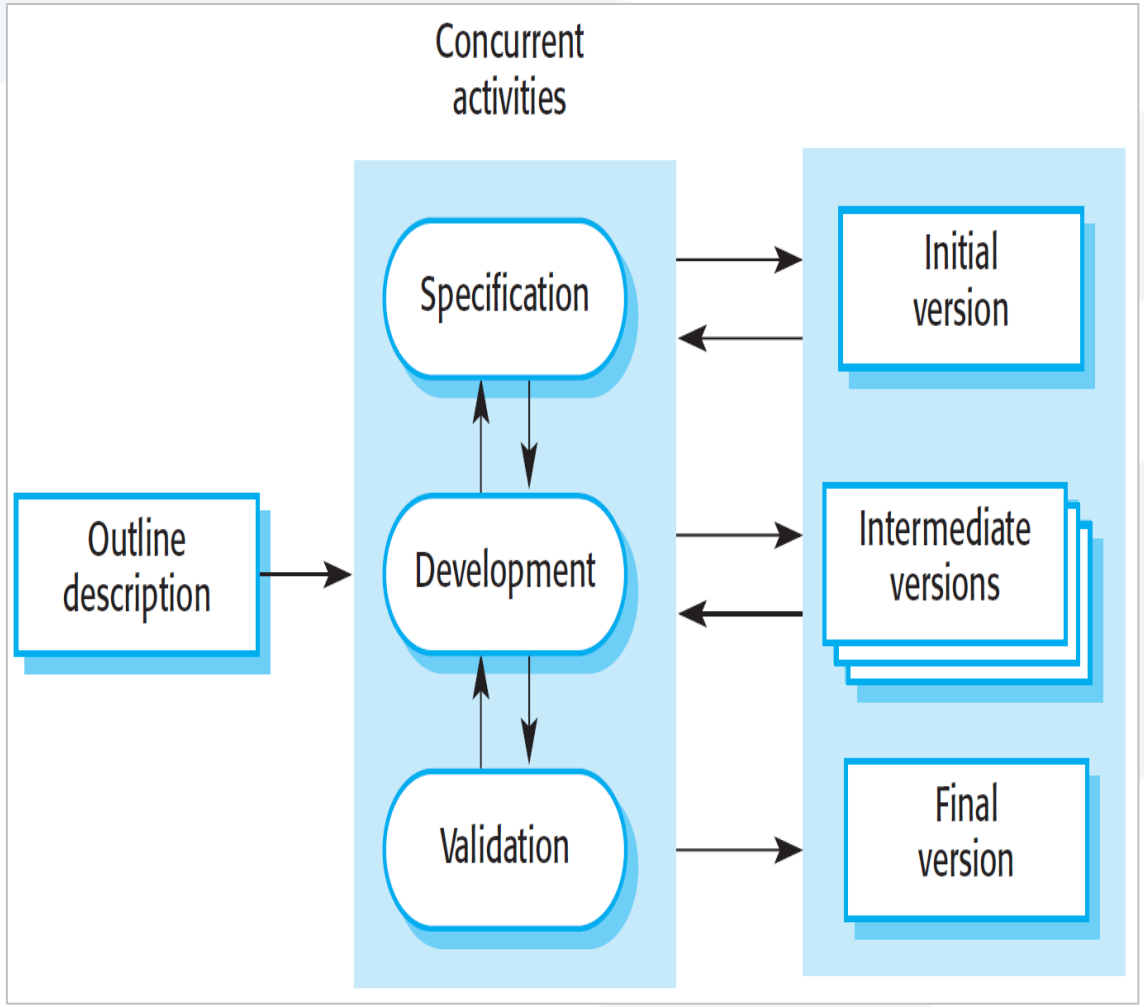
الإيجابيات:

- سهل الفهم: نموذج الشلال الكلاسيكي بسيط للغاية وسهل الفهم.
- مُعرّف بدقة: في نموذج الشلال الكلاسيكي، تكون كل مرحلة في النموذج مُعرّفة بوضوح.
- مُوثّق بدقة: العمليات والإجراءات والنتائج موثقة جيدًا.
- يُعزز العادات الجيدة: يُعزز نموذج الشلال الكلاسيكي العادات الجيدة مثل التعريف قبل التصميم والتصميم قبل البرمجة.

السلبيات:

- مرونة محدودة: نموذج الشلال هو نهج صارم وخطي لتطوير البرمجيات، مما يعني أنه غير مناسب للمشاريع ذات المتطلبات المتغيرة أو غير المؤكدة. بمجرد اكتمال مرحلة، يصعب إجراء تغييرات أو العودة إلى مرحلة سابقة. مشاركة محدودة لأصحاب المصلحة: نموذج الشلال هو نهج منظم ومتسلسل، مما يعني أن أصحاب المصلحة يشاركون عادةً في المراحل الأولى من المشروع (جمع المتطلبات وتحليلها
- تسليم المشروع لا يتم إلا بعد انتهاء كامل مراحل تطوير البرمجية(الوقت طويل من مرحلة الفكرة حتى الحصول على النظام) ، يُعاني النموذج من تأخر إنتاج أي برنامج قابل للاستخدام حتى مراحل متأخرة من دورة حياة المشروع.

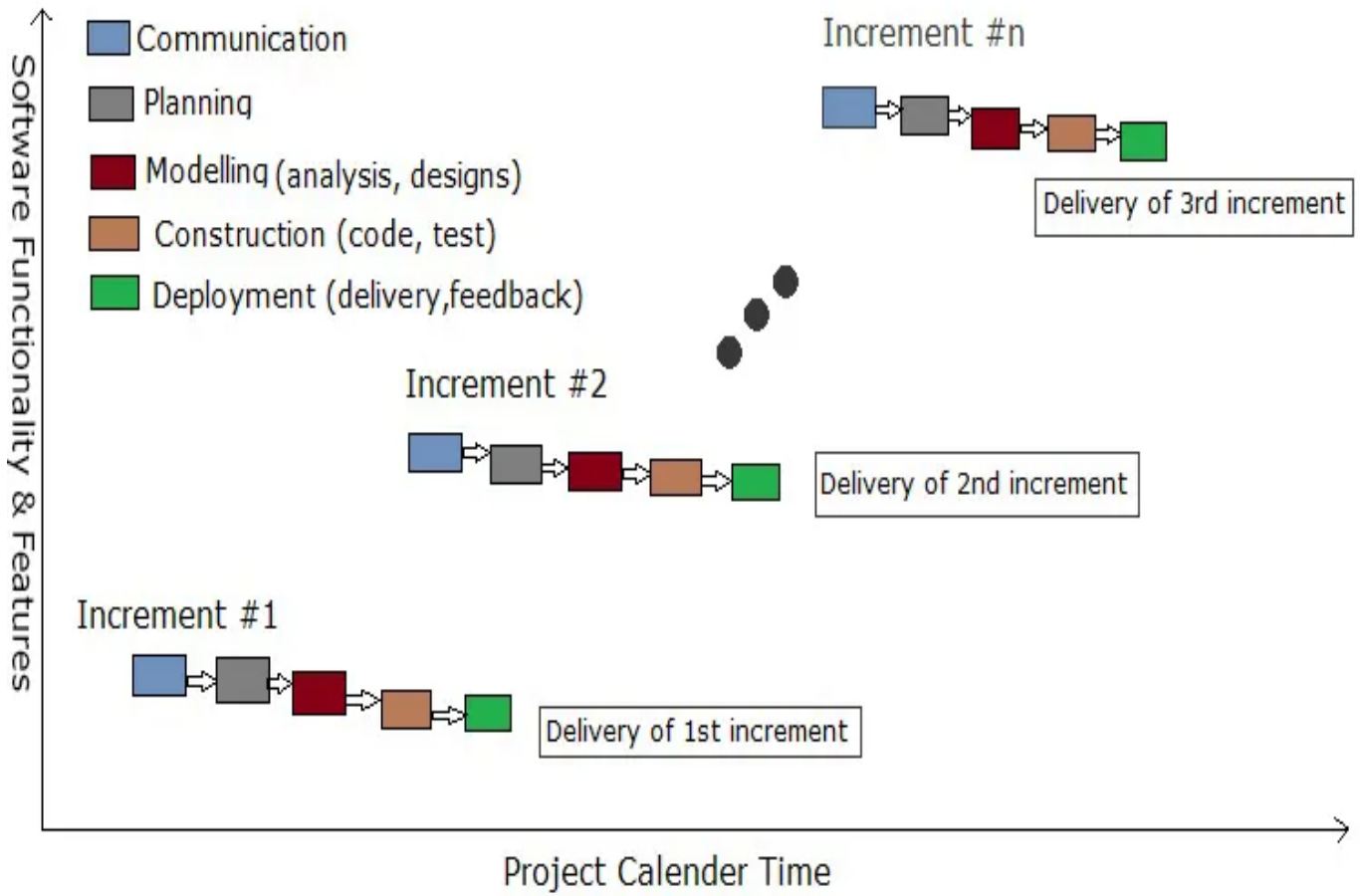
النموذج التزايدي Incremental model



- النوذج التزايدي هو أسلوب لتطوير البرمجيات، حيث يُبنى النظام تدريجيًا فبدلًا من تسليم النظام بأكمله دفعةً واحدة، يُطوَّر ويُسلَّم على أجزاء صغيرة تُسمى زيادات. كل زيادة تُكمِّل السابقة بإضافة وظائف جديدة، حتى يكتمل النظام بالكامل.
- يعتمد التطوير التزايدي على فكرة تطوير اصدار أولي (Initial version)، والحصول على تغذية راجعة (Feedback) من العملاء وتطوير البرمجيات من خلال عدة إصدارات (Versions or Increments) حتى يتم تطوير النظام المطلوب
- مع كل إصدار، يتم تقديم جزء من من الوظائف المطلوبة.
- يتم تحديد أولويات للمتطلبات ومن ثم يتم تضمين المتطلبات ذات الأولوية الأعلى في الإصدارات الأولى. تتوفر الوظائف الأساسية في بداية المشروع، مما يُتيح للمستخدمين البدء في استخدام النظام واختباره بسرعة
- تُجمع الملاحظات بعد تسليم كل جزء، مما يُساعد على تحسين الإصدار التالي من النظام.
- المرونة في التغييرات: يُمكن إضافة تغييرات أو ميزات جديدة بين مراحل التطوير، مما يجعل النموذج مرنًا لتلبية الاحتياجات المتغيرة.
- يفضل استخدام هذا النموذج في حال:
  - تغير المتطلبات متوقع بشكل مستمر أثناء عملية التطوير
  - مطلوب إصدار من المنتج البرمجي قابل للتسليم بسرعة
  - الفريق البرمجي غير متاح على كامل فترة المشروع البرمجي

النموذج التزايدي Incremental model

- **الإيجابيات**
  - تخفيض تكلفة تلبية متطلبات العملاء المتغيرة حيث يمكن لهذا النموذج التعامل مع تعديل او إضافة متطلبات في كل بداية كل دورة
  - الوقت القصير الذي يمكن للعملاء من الحصول على منتج ملموس ولكنه غير كامل، وبالتالي يمكّن العملاء من استخدام المنتج البرمجي في وقت أبكر مما هو ممكن في نموذج الشلال، وكذلك يكون من الأسهل الحصول على رأي العملاء على الإصدار الذي تم تسليمه
  - إمكانية العمل على أكثر من إصدار بشكل متوازي
- **السلبيات**
  - يمكن لبعض المتطلبات الجديدة ان تتناقض مع متطلبات قديمة قد تم تنجيزها بالفعل في النظام مما يضطرنا الى نسفها
  - يمكن ان ترتبط الخدمات ببعضها فيتم تقديم خدمة ولكنها غير مفيدة بدون خدمة أخرى مما يضطر العميل الى الانتظار دورة كاملة حتى يتم تنجيز هذه الخدمة وقد يعود هذا الى سوء في ترتيب الاولويات في النظام.



- إمكانية العمل على أكثر من إصدار بشكل متوازي.

- السلبيات:

- صعوبة قياس التقدم في إنجاز المشروع البرمجي.
- زيادة تكلفة ومدة المشروع وخصوصاً في حال توثيق كامل الإصدارات.
- ترهل البنية التركيبية للمنتج البرمجي بزيادة عدد الإصدارات.
- زيادة التكلفة بسبب الحاجة إلى إعادة تركيب وترتيب بنية المنتج البرمجي باستمرار حتى تصبح من الصعوبة القيام بإجراء التغييرات والتحسينات.

المراجع

- Al-khatib, B. et al. (2018). Essentials in Informatics Engineering (الأساسيات في الهندسة المعلوماتية). ISBN:978-0-7020-6064-9

- Sommerville, I. (2016). *Software Engineering (*10th ed.). ISBN:978-1-292-09613-1. Pearson Education.

- Pressman R. & Maxim B. (2015). Software Engineering: A Practitioner's Approach (8th ed.). ISBN:978-0-07-802212-8. McGraw-Hill.


HAVE YOU ANY
QUESTION