هندسة برمجيات ١

المحاضرة الثالثة

د. اناس ليلى

الفصل الدراسي الاول ٢٠٢٥ / ٢٠٢٦

Software Development Life Cycle (SDLC) models

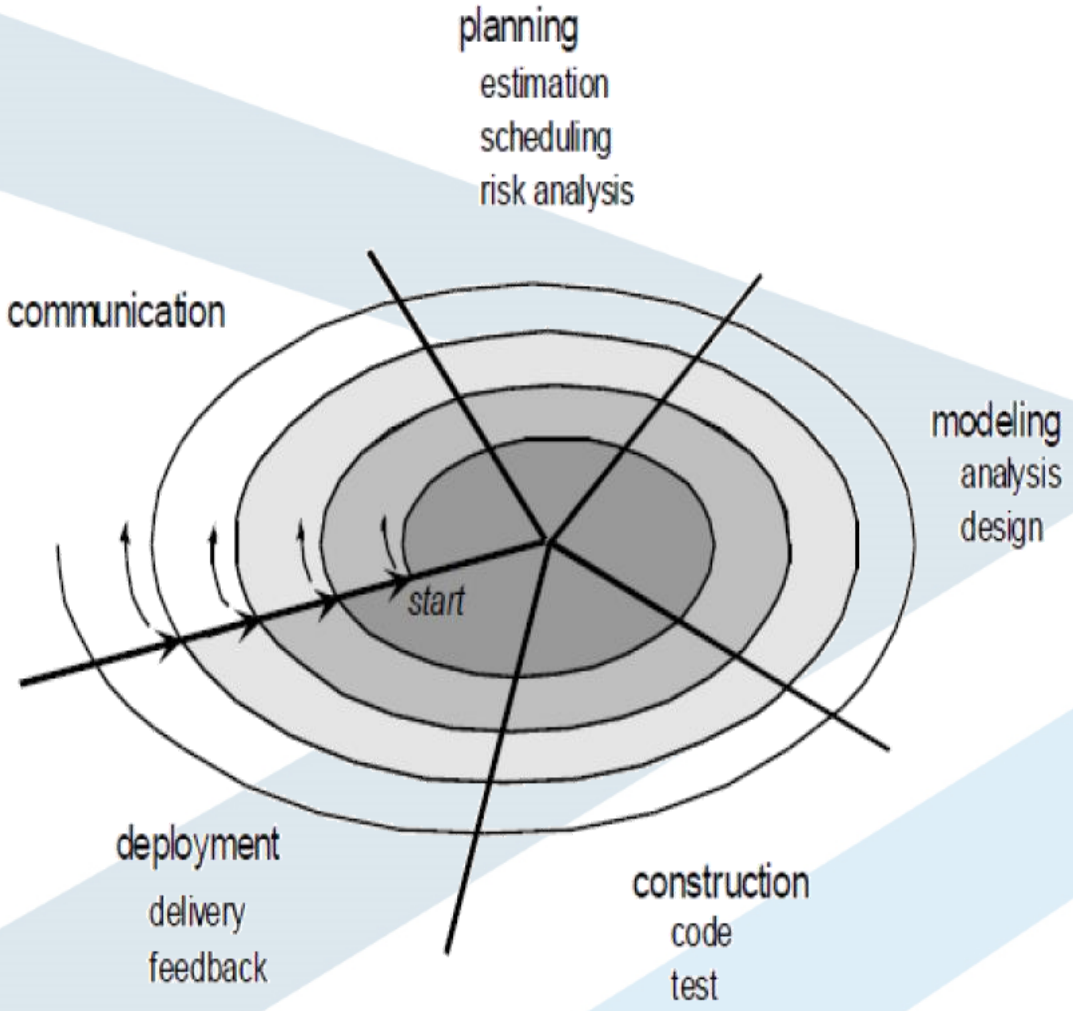
تُوجِّه نماذج دورة حياة تطوير البرمجيات (SDLC) عملية التطوير لضمان جودة عالية للبرمجيات. ويتم اختيار نماذج مختلفة لدورة حياة تطوير البرمجيات بناءً على الاحتياجات المحددة للمشروع والمؤسسة.

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

Spiral model النموذج الحلزوني

The Spiral Model is a risk-driven software development lifecycle (SDLC) model that combines iterative development with systematic risk management.

Developed by Barry Boehm in 1986, it consists of main phases repeated in cycles (spirals), with each iteration building upon the previous one.



- النموذج الحلزوني هو نموذج تطوير برمجيات يجمع بين النهج التزايدي incremental وبين إدارة المخاطر. اقترحه Barry Boehm في الثمانينيات ليعالج مشاكل النماذج التقليدية مثل الشلال عندما تكون المشاريع ذات مخاطر والمتطلبات غير واضحة أو المشروع كبير ومعقّد.
- النموذج يفترض أن المخاطر هي العامل الرئيسي لفشل المشروعات الكبيرة بالتالي يعالج المخاطر مبكراً قبل أن تكبر ويصبح إصلاحها مكلفاً، وهنا يتم تحديد المخاطر في كل دورة وهي تنتمي الى أصناف متعددة منها المخاطر التقنية، مخاطر التكلفة، مخاطر الجدولة الزمنية، مخاطر خارجية، ومخاطر تتعلق بالموارد حيث يتم تقييم هذه المخاطر ووضع الحلول التي يجب اتباعها في حال وقوع هذا الخطر.
- إذا أُحسن تطبيق هذا النموذج (بتخطيط واضح، تحليل مخاطر فعّال، ودورات زمنية مناسبة) فإن ذلك يقلل من المفاجآت المكلفة ويزيد فرص نجاح المشروع.
- المشروع البرمجي وفق هذا النموذج يُدار كـ **سلسلة من الحلقات Spirals** كل حلقة تمثل دورة تطوير صغيرة
- تزيد كل دورة من نضج الإصدار المسلم الى الزبون (يزوده بميزات جديدة أو تُحسّن التصميم) حتى الوصول الى المنتج النهائي.

متى نستخدم النموذج الحلزوني؟

- ❑مشاريع كبيرة أو معقّدة (نظم حكومية، بنوك، أنظمة طيران)
- ❑مشاريع تحتوي على مخاطر تقنية عالية (تقنيات جديدة، تكامل مع أنظمة قديمة)
- ❑عندما يكون التتبّع وإدارة المخاطر مهماً للعميل أو المتعاقدين

السلبيات

- ❖قد يكون مكلفا ومعقَّدا إدارياً.
- ❖يتطلب خبرة جيدة في تحليل وإدارة المخاطر.

مثال:

المنتج: ليكن لدينا نظام إلكتروني لإدارة وحجز مواعيد عيادة طبية مع بوابة للمرضى والطبيب وإدارة سجلات بسيطة.

المتطلبات الوظيفية: تمكين المرضى من حجز/إلغاء المواعيد، وإرسال تذكيرات، وتمكين الطاقم من إدارة المواعيد وسجلات المرضى./**المتطلبات غير الوظيفية: سرية البيانات وسرعة الاداء**

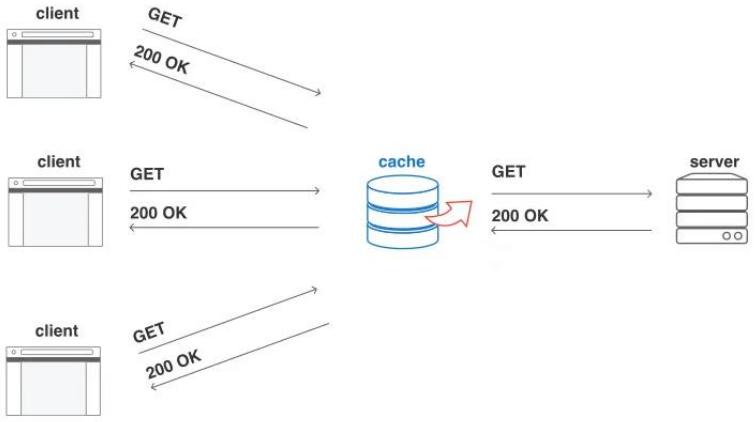
مبدئياً: واجهة ويب + واجهة موبايل خفيفة ، قاعدة بيانات مركزية، تكامل مع خدمة SMS/Email

لماذا الحلزوني مناسب هنا؟

- هناك مخاطر (قانونية/خصوصية، تكامل مع أنظمة اخرى SMS، قد يتم تحميل النظام بشكل زائد) وهذه المخاطر يجب تحليلها مبكراً.
- نريد إطلاق تدريجي Incremental Releases وتحسين مستمر بناءً على ملاحظات المستخدمين.

المخاطرة	الاحتمال	التأثير	الحلول
خرق بيانات مرضى	متوسط/عالي	عالٍ	تشفير، صلاحيات صارمة
تعطل مزود SMS	متوسط	متوسط	مزود بديل، آلية Retry، استخدام البريد الإلكتروني كخيار
أداء ضعيف عند الذروة	متوسط	عالي	اختبار زيادة تحميل مبكر، استخدام الكاش، تقسيم قاعدة البيانات DB، Autoscaling

```
def send_sms():
    retries = 3
    wait = 2  # ثانيتان
    for attempt in range(retries):
        success = sms_provider.send()
        if success:  return True
        time.sleep(wait)
        wait *= 2  # exponential backoff
    return False
```



سنوضح بعض الاليات المستخدمة في خطة التخفيف:

Retry: إعادة المحاولة تلقائياً عندما تفشل خدمة أو طلب فيقوم النظام بمحاولة تنفيذها مرّة أخرى بشكل تلقائي بفواصل زمنية محسوبة، لتحسين الاعتمادية والمتانة في النظام.

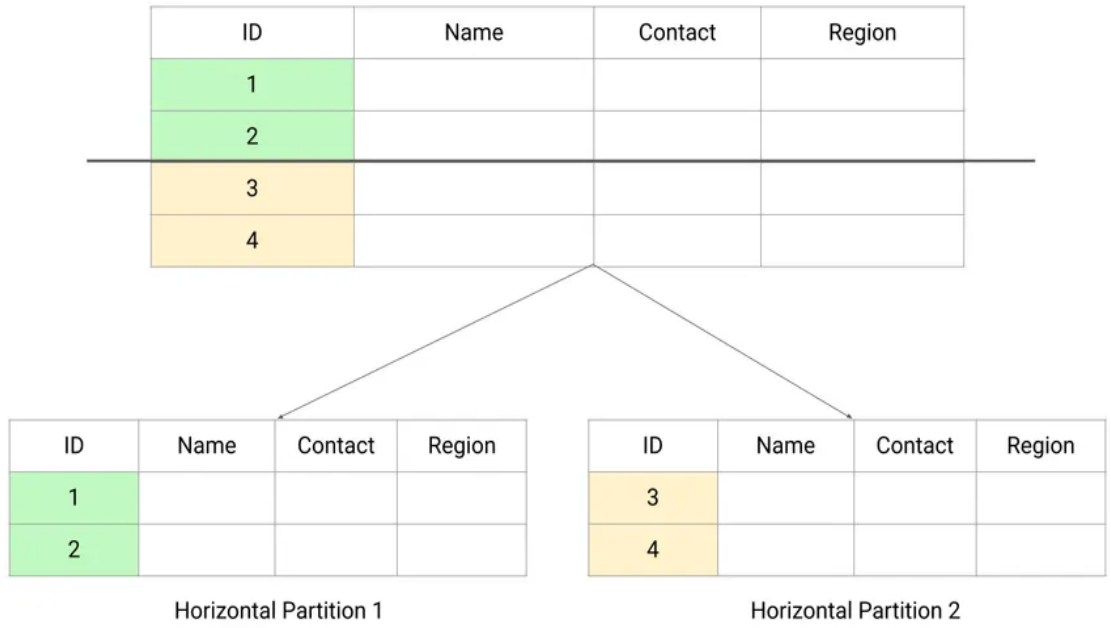
مثال: إذا حاول النظام إرسال رسالة SMS للمريض وفشل المزود (بسبب ضغط، انقطاع، أو خطأ مؤقت)، فإن النظام يعيد المحاولة تلقائياً.

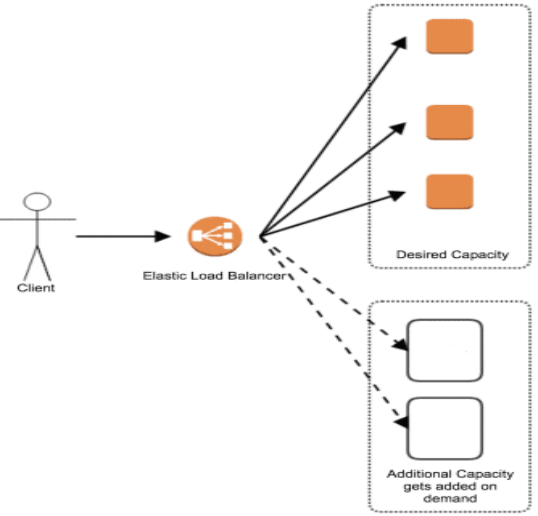
تسمى استراتيجية Retry الشائعة Exponential Backoff حيث أن كل محاولة تكون بعد فترة أطول من السابقة:١ ثانية٢ ثواني٤ ثواني٨ ثواني و عادة يكون هناك حد أقصى لعدد مرات المحاولة

الكاش: تخزين مؤقّت للبيانات الأكثر استخداما بحيث يمكن الوصول إليها **بسرعة أكبر** من الوصول لقاعدة البيانات الرئيسية.

نستخدمه لتسريع قراءة البيانات ولتقليل الضغط على قاعدة البيانات ولتحسين الأداء في أوقات الذروة

أمثلة على البيانات التي توضع في الكاش: نتائج استعلامات تتكرر كثيرا





Database Partitioning:

تقسيم قاعدة البيانات المقصود هنا هو توزيع البيانات على عدّة قواعد بيانات بدلاً من قاعدة واحدة. فعندما يكبر النظام وتزداد العمليات Queries تصبح قاعدة بيانات واحدة غير كافية.

لذلك يتم **تقسيم البيانات** إلى أجزاء منفصلة: حسب المستخدم أو حسب النوع أو حسب التاريخ أو حسب عيادة / طبيب / منطقة

مما يقلل الضغط على قاعدة البيانات ويسرع الأداء ويمكن من التوسّع بسهولة.

Autoscaling زيادة أو تقليل عدد الخوادم تلقائياً حسب الحمل حيث يتم مراقبة عدد الطلبات في الثانية ونسبة استخدام CPU والذاكرة وعندما يزيد الحمل تضاف خوادم إضافية تلقائياً وعندما يقل الحمل تقلل عدد الخوادم.

أي يصبح التوسع أوتوماتيكيا عند الضغط وبالتالي لا ينهار الأداء وقت الذروة.

Prototyping model

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development.

It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

نموذج النموذج الأولي : يعتمد على بناء نسخة أولية Prototype سريعة وغير مكتملة من النظام بهدف فهم المتطلبات بدقة وذلك عندما لا يستطيع الزبون التعبير عن متطلباته حيث نقوم بتصميم نموذج اولي يتألف من واجهات دخل وخرج **غير فعالة** تحفز الزبون وتزيد قدرته على توصيف متطلباته، كما يستخدم لتقليل الغموض(تطبيق جديد ذو فكرة جديدة غير مطبقة سابقا و لاختبار الأفكار مبكرا، استقبال ملاحظات المستخدمين بسرعة و تحسين التصميم قبل البناء الفعلي.

ويعد هذا النموذج مناسبا جدا عندما تكون المتطلبات غير واضحة.

لماذا نقد نحتاج نموذج أولي في كثير من المشاريع وماهي الفكرة الأساسية منه؟

قد لا يكون لدى العميل فكرة واضحة عن الشكل النهائي للبرمجية التي يطلب تطويرها، أو يعرف ما يريد لكن لا يستطيع التعبير عنه، أو توجد قرارات تصميمية غير محسومة بعد.

ان الـ Prototype يجعل العميل يرى مبكرا شيئا ملموساً يشبه البرمجية التي يريدها فيصبح قادرا على تحديد المطلوب، وقد يطلب تعديل الواجهات ويقترح تحسينات قبل أن نصرف وقتاً وجهداً كبيراً في البرمجة الحقيقية.

يبني الفريق **نموذجاً** للبرمجية (واجهات فقط أو منطق جزئي) يقدمونه للزبون ويحصلون من خلاله على المتطلبات بدقة

النموذج الأولي هنا لا يعني أن المشروع انتهى—بل فقط فُهِم بطريقة صحيحة.

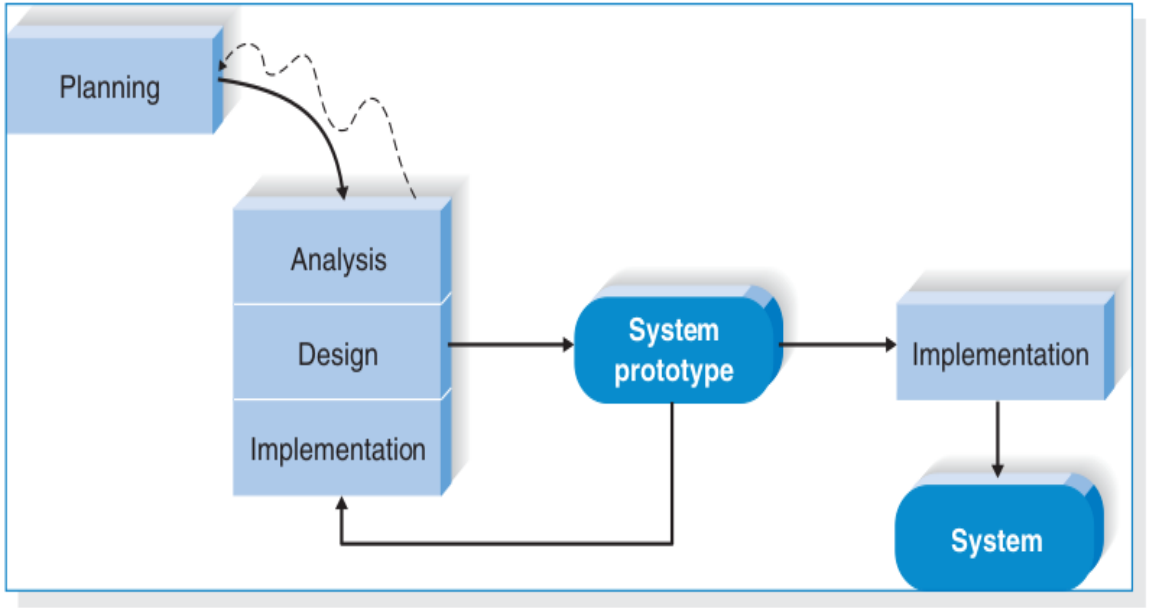
بعد الموافقة قد يتم بناء النظام الحقيقي وهنا نميز بين نوعين من النماذج الأولية (الاستكشافي والتطويري) ففي الاستكشافي يتم رمي النموذج الذي تم بناؤه ويتم بناء النظام **من الصفر وفي التطويري يتم بناء النظام بالاعتماد على النموذج الاولي** الذي تم بناؤه.

أنواع النماذج الأولية (Types of Prototypes)

١- النموذج الأولي التطويري (Evolutionary Prototype):

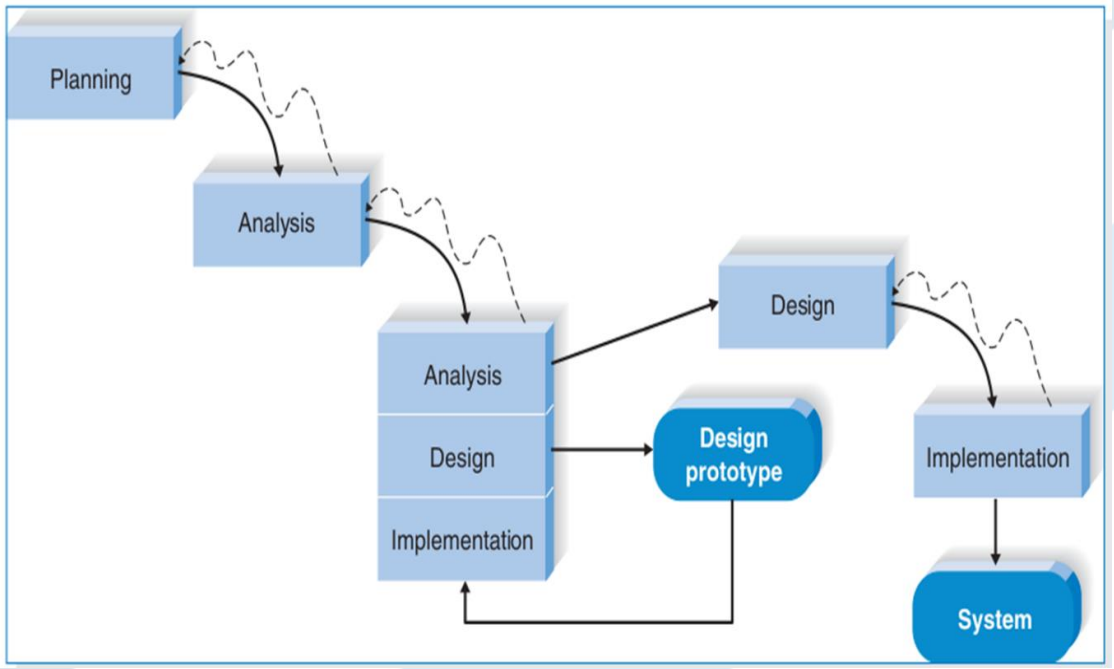
يستخدم هذا النموذج كأساس ويطور للوصول الى المنج النهائي

مثال: تطبيق حجز مواعيد يتم تطويره تدريجيا إلى أن يصبح المنتج الأساسي.



٢- النموذج الأولي الاستكشافي (Throwaway / Exploratory):

الهدف منه فهم المتطلبات فقط ثم يتم التخلص منه بعد تحقيق الهدف منه ولا يتحول إلى جزء من المنتج النهائي. مثال: رسم واجهات على HTML بسيط فقط لمعرفة ما يريده العميل.



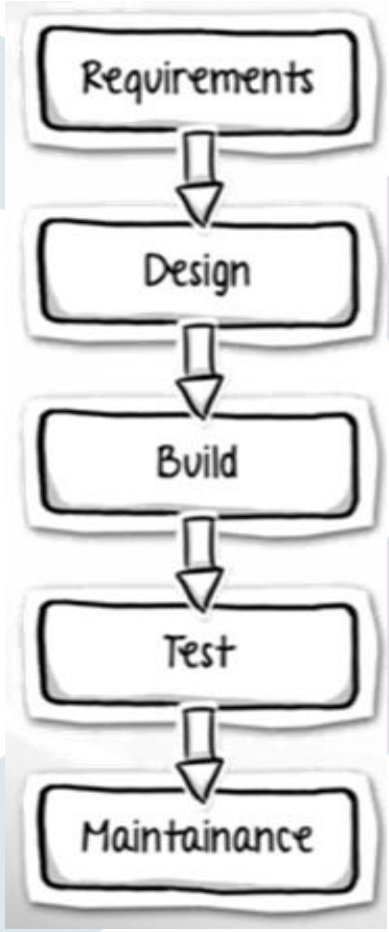
SDLC V-Model

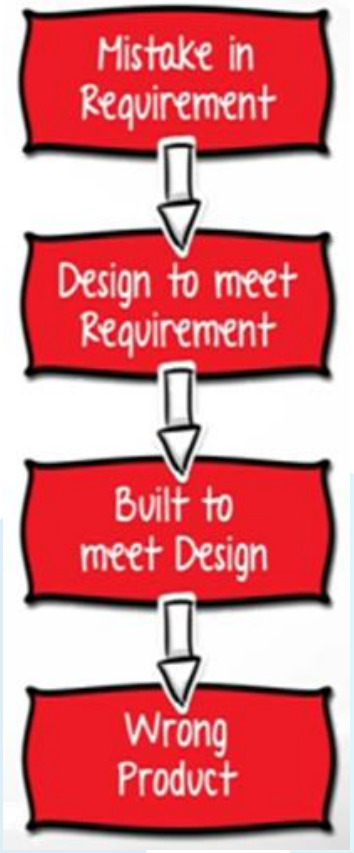
The V-Model stands as one of the most systematic and quality-focused Software Development Life Cycle (SDLC) methodologies. It is also known as the Verification and Validation model.

It extends the traditional waterfall model by emphasizing verification and validation at each phase of development.

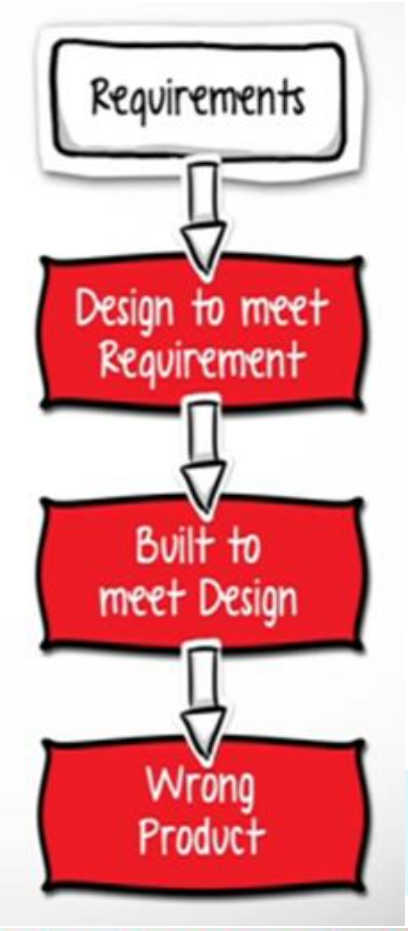
This model focuses on verification and validation activities early in the life cycle thereby enhancing the probability of building an error-free and good quality product

WATERFALL MODEL





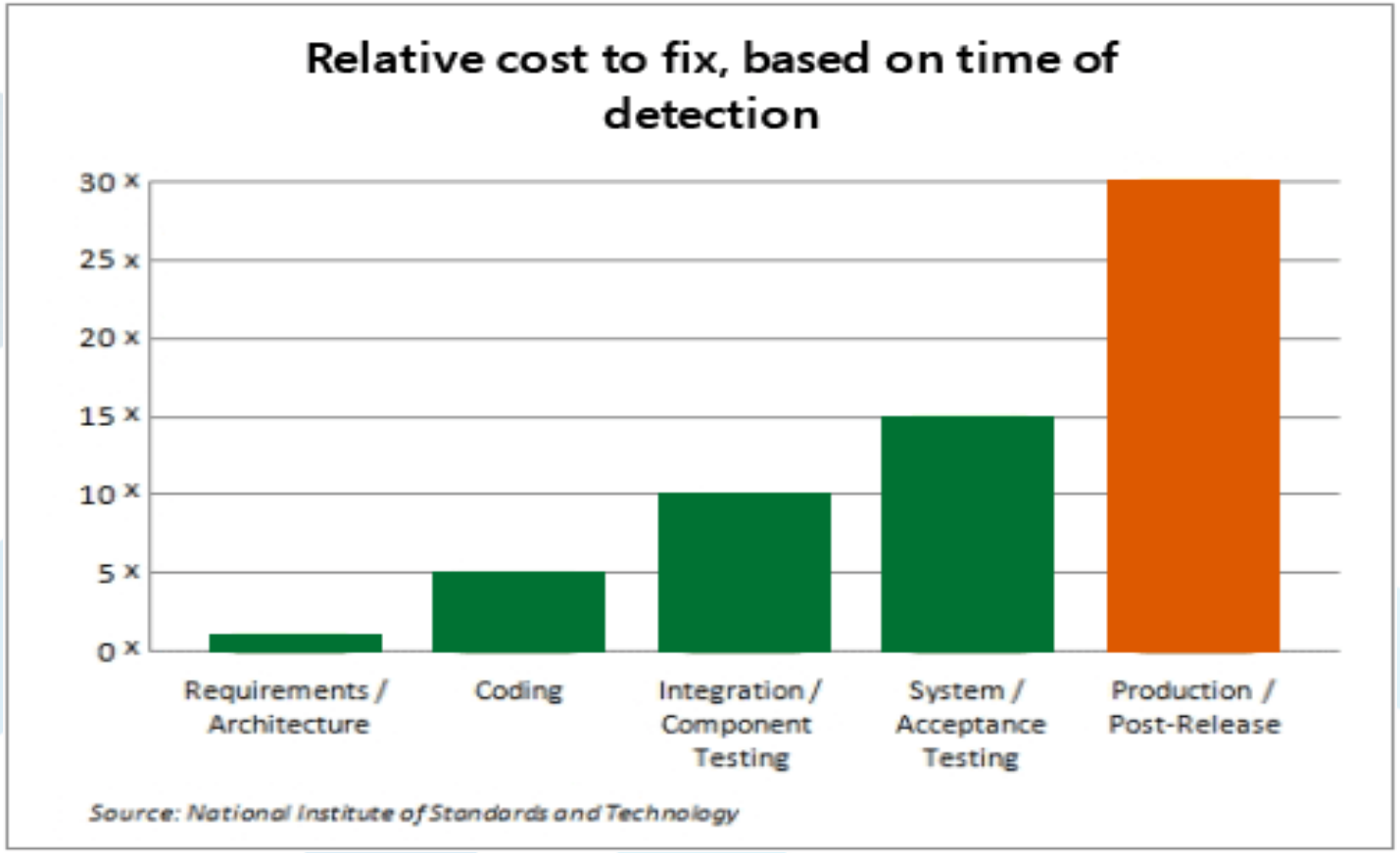
You will have start afresh with project



Re-Design to correct defects

أظهرت تقييمات آلاف المشاريع أن العيوب التي تم إدخالها أثناء المتطلبات والتصميم تشكل ما يقرب من نصف العدد الإجمالي للعيوب.
تزداد تكاليف إصلاح العيب كلما تقدمنا أكثر في دورة حياة تطوير المنتج البرمجي. كلما تم اكتشاف العيب مبكرًا في دورة الحياة، قلّت تكلفة إصلاحه.

The earlier in life cycle a defect is detected, the cheaper it is to fix it.



تم تطوير نموذج الاختبار V حيث يوجد لكل مرحلة في دورة حياة التطوير مرحلة اختبار مقابلة

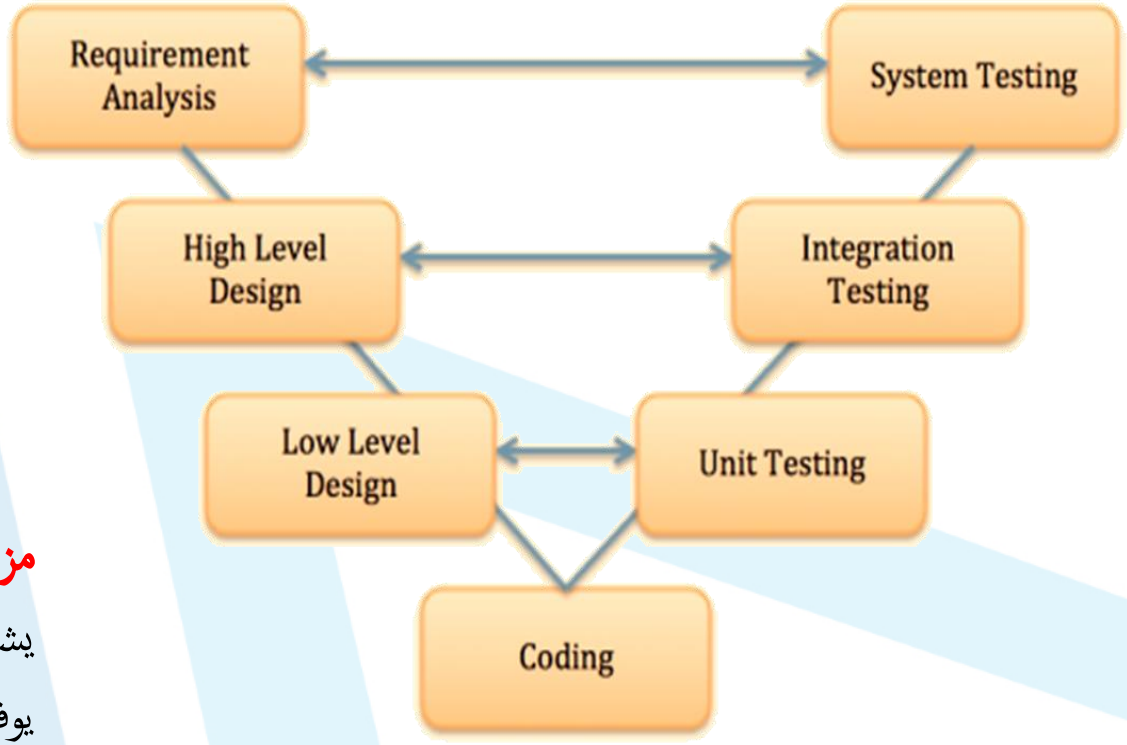
The V-Model consists of two main phases:

Verification Phase of V-Model (left side of V)

Validation Phase of V-Model (right side of V)

يتكون النموذج V من مرحلتين رئيسيتين: مرحلة **Verification** وهي الجانب الأيسر من النموذج (V) وهي تركز على تحليل وتصميم النظام قبل بدء البرمجة.

أما مرحلة **Validation فهي تتحقق من** أن البرنامج المطوّر يتوافق مع متطلبات الزبون وتوقعاته.



V Model

مزايا النموذج V

يشجع على الكشف المبكر عن العيوب، مما يقلل التكلفة وإعادة العمل.

يوفر هيكلًا واضحًا يربط المتطلبات بأنشطة الاختبار.

يضمن جودة عالية للمخرجات من خلال عمليات تحقق دقيقة.

متى يُفضل استخدام هذا النموذج؟ يقدم الجدول مقارنة بين نموذج الشلال وV-Model

الأفضلية للمشاريع	علاقة الاختبار	مرونة التغيير	مراحل واضحة؟	النموذج
الصغيرة/الثابتة	بعد التطوير	منخفضة	نعم	الشلال
الحرجة/الثابتة	مع كل مرحلة	منخفضة	نعم	V-Model

هذه النماذج ما زالت مستخدمة في مشاريع برمجية كثيرة، خاصة عند الحاجة لتخطيط محكم وتوثيق شامل ضمن بيئة تطوير قليلة التغيير.

مبادئ النموذج V

يعتمد النموذج V على عدة مبادئ أساسية:

١. من الكبير إلى الصغير: تتطور المتطلبات من عالية المستوى إلى تفصيلية، ويعكس الاختبار ذلك.

٢. قابلية التتبع: يرتبط كل متطلب بحالة اختبار مقابلة.

٣. الاختبار المبكر: تبدأ أنشطة الاختبار بمجرد تحديد المتطلبات.

٤. التركيز على التوثيق: تُنتج كل مرحلة مخرجات قابلة للمراجعة والرجوع إليها.

٥. قابلية التوسع: ينطبق على المشاريع الصغيرة والكبيرة ذات المتطلبات المستقرة.

Reuse-Based Software Development Model (Reuse Oriented Model)

نموذج تطوير البرمجيات المعتمد على إعادة الاستخدام هو نهج يركز على إنشاء أنظمة برمجية من خلال دمج المكونات الموجودة والمطورة مسبقًا بدلاً من بنائها من الصفر

نموذج تطوير البرمجيات المعتمد على إعادة الاستخدام هو أحد النماذج المهمة في هندسة البرمجيات الحديثة. يعتمد هذا النموذج على مبدأ مفاده بأنه يمكن تسريع تطوير البرمجيات وجعلها أكثر كفاءة من خلال الاستفادة من المكوّنات الجاهزة (Components, Libraries, Services, Code Modules) القابلة لإعادة الاستخدام بدلًا من تطوير كل شيء من الصفر فبدلاً من بناء ١٠٠% من النظام، نعيد استخدام ما يمكن إعادة استخدامه، ونقوم فقط بتطوير الأجزاء الجديدة. هذا النموذج يقلل الوقت والتكلفة ويزيد الجودة (لأن المكوّنات المختبرة تكون أقل عرضة للأخطاء)

أنواع إعادة الاستخدام (Reuse Types)

١-إعادة استخدام مباشرة Black-box Reuse: استخدام المكوّن كما هو بدون تغيير. مثال: استدعاء مكتبة Pandas في Python

٢- اعادة استخدام مع تعديل White-box Reuse: نأخذ المكوّن ونعدّله ليتناسب مع النظام. مثال: أخذ كود جاهز مثلا من GitHub وإجراء تعديلات عليه

٣- إعادة استخدام تصميم مسبق Design Reuse: إعادة استخدام تصميمات أو نماذج UML, Architecture ، Design Pattern

٤-إعادة استخدام مبني على الخدمات (Service-Oriented Reuse): استخدام خدمات جاهزة مثل: APIs، Web Services ، Cloud Functions

Advantages:

- Reduces development effort and costs.
- Accelerates software delivery.

عيوب نموذج إعادة الاستخدام

- ✗ البحث عن المكوّن المناسب قد يأخذ وقتاً
- ✗ التوافق بين المكونات قد يكون صعباً
- ✗ مشاكل الملكية الفكرية والترخيص/licensing
- ✗ التعديل على المكونات الجاهزة قد يكلف كثيراً

أمثلة على إعادة الاستخدام

١) في الويب

استخدام:

- Bootstrap
- React Components

٢) في تطبيقات الهاتف

استخدام مكتبات جاهزة مثل Firebase Authentication بدلاً من بناء نظام تسجيل دخول.

React هي مكتبة JavaScript مفتوحة المصدر تم تطويرها لبناء واجهات المستخدم، وهي مكتبة مرنة تجعل من السهل إنشاء واجهات مستخدم تفاعلية باستخدام نهج قائم على المكونات (component-based approach) والمكونات هي أجزاء صغيرة من الكود يمكن إعادة استخدامها لإنشاء تطبيقات ويب.

ما هو بوتستراب؟

Bootstrap عبارة عن Library مجانية تستخدم لتسهيل عملية تصميم صفحات الويب على المطور حيث يوفر له كلاسات CSS جاهزة يمكنه استخدامها لإظهار العناصر (Elements) التي نضيفها في الصفحات بشكل جميل جداً و متجاوب (Responsive) مع حجم الصفحة مثل الأزرار ، مربعات النص ، القوائم المنسدلة و غيرها من الأشياء المساعدة مثل رسائل التنبيهات التي نظهرها للمستخدم.

إذاً التصميم الذي تعتمد على بوتستراب في بنائه، سيظهر بشكل جميل على جهاز المستخدم سواء كان يستخدم هاتف، حاسوب أو تابلت أو أي جهاز آخر

فايربيز (Firebase) هي منصة تطوير تطبيقات أطلقتها شركة **جوجل**، وتُستخدم لبناء وتطوير تطبيقات الأجهزة الذكية وتطبيقات الويب بشكل سريع وآمن. توفر فايربيز مجموعة من الأدوات والخدمات السحابية التي تُستخدم مباشرة من تطبيق المستخدم دون الحاجة إلى بناء خدمات الـ (Backend) بشكل تقليدي.

تشمل الخدمات الرئيسية في فايربيز:

- **المصادقة (Authentication)**: تسجيل الدخول عبر البريد الإلكتروني، كلمات المرور، أو عبر وسائل التواصل الاجتماعي مثل جوجل وفيسبوك.
- **قواعد البيانات (Firestore & Realtime Database)**: تخزين البيانات وتحديثها فورًا عبر جميع المستخدمين.
- **تخزين الملفات (Cloud Storage)**: لرفع وتخزين الصور، الفيديوهات، والملفات الأخرى.
- **التحليلات (Analytics)**: تتبع سلوك المستخدمين داخل التطبيق.
- **الإشعارات (Cloud Messaging)**: إرسال إشعارات فورية إلى المستخدمين.
- **استضافة التطبيقات (Hosting)**: لنشر تطبيقات الويب بسرعة.

تُستخدم فايربيز بكثرة في تطبيقات الأندرويد، iOS، والويب، وتدعم لغات برمجة متعددة مثل JavaScript وFlutter وUnity.

ملخص لأشهر النماذج تطوير البرمجيات

Waterfall Model النموذج الشلالي

١. تسلسل صارم من المراحل: التحليل ثم التصميم ثم التنفيذ ثم الاختبار ثم الصيانة.
٢. لا يُنتقل إلى مرحلة قبل إكمال السابقة.
٣. ✓ سهل الإدارة – مناسب للمشاريع الصغيرة الواضحة المتطلبات.
٤. ✗ ضعيف في التعامل مع التغييرات.

١. Incremental Model النموذج التزايدي

١. يتم بناء النظام على شكل إصدارات متتابعة، كل إصدار يضيف وظيفة جديدة الى الإصدار السابق.
٢. ✓ يوفر منتجا أوليا بسرعة، وقابل للتطوير التدريجي.
٣. ✗ يتطلب تخطيطا دقيقا لكل تزايد.

٢. Spiral Model النموذج الحلزوني

١. يجمع بين التصميم التزايدي وإدارة المخاطر.
٢. المشروع يمر بعدة دورات
٣. ✓ جيد للمشاريع الكبيرة ذات المخاطر العالية.
٤. ✗ معقد ويتطلب خبرة كبيرة.

٣. V-Model النموذج V

١. تطوير يربط كل مرحلة من مراحل التطوير بمرحلة اختبار مقابلة.
٢. ✓ جودة عالية للاختبار.
٣. ✗ لا يناسب المشاريع ذات المتطلبات المتغيرة.

النموذج	الفكرة الأساسية
الشلال	تطوير خطي من الصفر
V-Model	تركيز على الاختبار
Agile	تطوير تكراري مرن
Prototype	بناء نموذج أولي
Spiral	إدارة مخاطر
Reuse-Based	استخدام مكونات جاهزة بدل البرمجة من الصفر

جَـامعة
المَـنارة
MANARA UNIVERSITY