# هندسة البرمجيات

## المحاضرة الاولى

## مقدمة عن هندسة البرمجيات

مدرس المادة: م.م نور حسن حسون

# المقدمة

## البرمجيات ( Software)

- الحاسب الآلي بدون برمجيات كإنسان بلا روح ، كما أن التطور الذي يحدث في أجهزة الحاسب الآلي ومكوناته يصاحبه أيضا تطور وتحديث دائم في عالم البرمجيات.

- والبرمجيات بصفة عامة هي عبارة عن مجموعة من الأوامر المرتبة منطقيا ، ويتم تنفيذها بواسطة وحدة المعالجة المركزية للحاسب الآلي ، ويختلف مستوى ونوع البرمجيات طبقا لعلاقاته وقربه من الحاسب الآلي من ناحية ، أو من قربه وعلاقته بالمستخدم من ناحية أخرى.

- فنجد أن نظام التشغيل Operting Systems بشكله الأولي هو الملتصق مباشرة بوحدة المعالجة المركزية CPUبينما نجد على الطرف الآخر ، التطبيقات البرمجية Applications هي الأكثر قربا وسهولة بالنسبة للمستخدم.

# المقدمة

## أنواع المنتجات البرمجية؟

- **برمجيات عامة شاملة(Generic)** : وهي نظم مستقلة تنتج بواسطة شركات وتباع في السوق لأي عميل، وأحياناً يطلق عليها أسم البرمجيات المغلفة مثل قواعد البيانات ومعالجات النصوص(word) وحزم الرسوم(Paint) .

- **برمجيات جاهزة (تفصيل أو مخصصة Customized)** : وهي نظم مخصصة لعميل معين يطلب تجهيزها، ويتم تطويرها بواسطة شركة أو مطور خصيصاً لهذا العميل ومنها أنظمة التحكم في المعدات الإلكترونية والآلآت والنظم الخاصة بأعمال معينة.

# المقدمة

## انواع البرمجيات:

- هناك ثلاثة أنواع من البرمجيات هي برمجيات:

١. برمجيات النظم (System Software)

يتولى هذا النوع من البرمجيات العديد من تفاصيل إدارة نظام الحاسوب، مثل نظم التشغيل operating system، معالجة اللغات , Compiler ولغات البرمجة Programing Language

٢. البرمجيات التطبيقية (Application Software)

هذه البرمجيات تطوع الحاسوب من أجل تنفيذ وظائف مفيدة وخاصة مثل معالجة الحسابات باستخدام برنامج حسابات متخصص وإدارة المخازن وجدولة المواد الدراسية وغيرها من البرمجيات المتخصصة في إدارة الأعمال المختلفة ويتم شراء هذه البرمجيات حسب الطلب من شركات الحاسوب المعنية بالبرمجة .

# المقدمة

٣. برمجيات الأغراض العامة (General Software):

وهي البرمجيات التي يستطيع أي شخص أن يستخدمها ومن برمجيات الأغراض العامة برامج معالج النصوص مثل Word، الجداول الإلكترونية Excel و حزم إدارة قواعد البيانات مثل Access التي تساعد في تنظيم واسترجاع.

# المقدمة

## عمليات البرمجيات Software Processes

هي مجموعة من الأنشطة التي تهدف إلى تنمية وتطوير البرمجيات. الأنشطة الرئيسية هي في عمليات البرمجية:

١. المواصفات Specifications: ما الذي يجب على النظام أن يفعله، وما هي قيود تطويره.

٢. التجهيز والتطوير Development : إنتاج نظام البرمجيات، أي أن نتيجة برمجيات يجب أن تحقق المواصفات.

٣. التحقق او التثبت Validation : فحص واختبار أن البرمجيات المنتجة تحقق المواصفات التي طُلِبَة من قبل العميل.

٤. تقييم البرمجيات Evolution : تغيير وتطوير البرمجيات استجابة للتغييرات الطارئة.

# المقدمة

## دورة حياة تطوير البرمجيات Software Lifecycle:

- عملية بناء أي منتج تمر بعدة مراحل يطلق علىها عادة "دورة الحياة " تتضمن المراحل التالية:

١. تحديد وتعريف المتطلبات Requirements analysis and definition .
٢. تصميم النظام System design.
٣. تصميم البرنامج Program design.
٤. كتابة البرنامج (تطويره) Program implementation.
٥. أختبار وحدات البرنامج Unit testing.
٦. أختبار النظام system testing.
٧. تسليم النظام system delivery.
٨. الصيا نة maintenance

# المقدمة

## خصائص البرمجيات الجيدة:

- يمكن وضع تعميم لخصائص البرمجيات الجيدة بحيث تحقق المواصفات التالية للبرمجيات الجيدة التى تسعى الغالبية العظمى من طرق هندسة البرمجيات والأدوات والتقنيات للمساعدة في إنتاج برمجيات تحقق هذه الخصائص وهي :

١. **قابلية الصيانة Maintainability:** يجب كتابة البرمجيات بطريقة تلبى احتياجات التغيير الى يحتاجها المستخدم، وهي خاصية حرجه لأن تغييرات البرمجيات نتاح حتمي لتغير بيئة العمل.

٢. **قابلية الاعتمادDependability:** تتضمن قابلية اعتمادي البرمجيات في نطاق من الخصائص تحتوى على الاعتماد عليها Reliabilityوالتأمين Security والأمان Safety، فالبرمجيات التى يعتمد عليها لا تسبب ضرراً أو تلفاً فيزيائياً او اقتصادياً في حالة حدوث انهيار النظام.

# المقدمة

## خصائص البرمجيات الجيدة:

٣. **الكفاءة Efficiency**: لا يجب على البرمجيات إهدار موارد النظام System Resources مثل:

- الذاكرة و دورات المعالج Processor Cycles، وبهذا تتضمن الكفاءة .
- الاستجابة . Responsiveness
- وقت المعالجة . Processing Time

٤. **قابلة الاستخدام Usability** : يجب أن تكون البرمجيات قابلة للاستعمال بدون مجهود غير مستحق للمستخدمين المصمم لهم النظام ، وهذا يعنى ضرورة وجود واجهة مستخدم (user interface) مناسبة مع المستندات التي تشمل على التوثيق الكافية.

# المقدمة

## هندسة البرمجيات Software engineering

- هي فرع من فروع المعلوماتية يهدف الى تطوير مجموعة أسس وقواعد تهدف إلي تحسين طرق تصميم وتطوير البرامج علي جميع المستويات وذلك بطريقه تلبي احتياجات المستخدمين .

- وهندسة البرمجيات لا تهتم بكتابة البرنامج نفسه أي بكتابة شفرته بل تحاول تحسين عملية تطوير وصنع البرنامج ابتداء من المواصفات التي يضعها المحترف وانتهاء عند مشكلة صيانة البرنامج أو توسعته

- كما انها تقوم علي دراسة احتياجات المستخدم وتصميم البرنامج المناسب لها قبل كتابة شفرته، وهناك العديد من الجوانب كالقدرة علي تطوير البرنامج بسهولة لاحقا، أو السرعة، أ و إمكانية إضافة ملحقات له بشكل ديناميكي

# مفهوم هندسة البرمجيات

- كان الاستخدام الرسمي الاول لهذا المصطلح في مؤتمر عقد من قبل اللجنة العلمية في منظمة حلف شمال الاطلسي عام ١٩٦٨ حول البرمجيات.

- عقد المؤتمر لمعالجة ما يعرف « ازمة البرمجيات» والتي ظهرت بسبب استخدام الوسائل التقليدية في بناء البرمجيات مما ادى الى ظهور برمجيات تحتاج الى وقت كبير لتطويرها.

- المكونات البرمجية ( software) شيء غير ملموس الى حد ما بالمقارنة مع المنتجات الاخرى كالمكونات المادية (Hardware).

- تمثل المكونات البرمجية سلسلة من الاف او ملاين الاوامر التي تطلب من الحاسوب اجراء عمليات معينة مثل عرض المعلومات او اجراء الحسابات او تخزين البيانات.

# مفهوم هندسة البرمجيات

- هندسة البرمجيات (Software Engineering) فهي فرع من فروع الهندسة يقوم على مجموعة اسس وقواعد تهدف الى تصميم وتطوير البرامج بوفرة ونوعية عالية تلبي احتياجات المستخدمين.

- هذا الفرع من الهندسة يتميز بانه لا يحتاج الى راس مال كبير وبالتالي الخسارة فيه تكون قليلة على عكس بقية الفروع من الهندسة.

- كما انه لا يكفي لا يجاد برمجيات متكاملة وجيدة بالاعتماد على عمل شخص واحد وانما يتطلب ذلك فريقا من المهندسين الاكفاء.

-

# مفهوم هندسة البرمجيات

## علاقة هندسة البرمجيات بعلوم الحاسوب

- علوم الحاسوب: يقوم بالتركيز على المكونات المادية، انظمة التشغيل، لغات البرمجة، المترجمات (Compiler).

- هندسة البرمجيات: هو تخصص يستخدم تكنولوجيا الحاسوب والبرمجيات كأدوات لحل مشكلة معينة.

- يمكن اخذ مثال بسيط يبين الفرق بين هندسة البرمجيات وعلوم الحاسوب في علم الكيماء واستخدامه في حل المشاكل التي نقابلها في حياتنا اليومية:

# مفهوم هندسة البرمجيات

## علاقة هندسة البرمجيات بعلوم الحاسوب

- يهتم الكيمائي بدراسة المواد الكيميائية ( تركيبها، تفاعلاتها، والنظريات التي تحكم سلوكها)

- بينما المهندس الكيميائي يستخدم النتائج التي توصل اليها الكيميائي لحل المشاكل التي يطلب منه ايجاد حل لها.

- اي انها من وجهة نظر الكيميائي هو موضوع الدراسة بحد ذاتها. بينما من وجهة نظر المهندس الكيميائي هي اداة Tool تستخدم لإيجاد حلول لمشكلة عامة ( قد لا تكون مشكلة ذات طبيعة كيميائية بحد ذاتها). الشكل التالي يبين العلاقة بين هندسة البرمجيات وعلوم الحاسوب.

# مفهوم هندسة البرمجيات

## علاقة هندسة البرمجيات بعلوم الحاسوب

# الفرق بين البرمجة وهندسة البرمجيات

- في البرمجة تعتبر عملية كتابة الكود هي اهم عملية في بناء البرنامج بغض النظر عن الجدوى من البرنامج او امكانية قبول المستخدم له او حتى قابلية التطوير.

- في حين ان هندسة البرمجيات تعمل على بناء النظام البرمجي كمشروع متكامل و دراسته من كافة الجوانب: البناء البرمجي، الدعم الفني والصيانة، التسويق والمبيعات، التطوير والتدريب على استخدامه.

# مهندس البرمجيات ( Software Engineer)

- مهندس البرمجيات هو الشخص الذي يقع على عاتقه تطوير المنتجات البرمجية التي تباع للزبائن او التي يحتاج اليها العملاء او الزبائن. لذلك يجب عليه ان يتبنى اسلوبا منظما ونظاميا في عمله لكي يحقق الاهداف المرجوة من استخدام علم هندسة البرمجيات.

- كما يتحتم عليه استخدام الادوات المناسبة ( **كاختيار لغة برمجية مناسبة من لغات البرمجة عالية المستوى)** والتقنيات الضرورية **(كالخوارزميات مختلفة الاغراض)** ويكون ذلك بالاعتماد على نوع المشكلة التي يقوم بحلها وقيود التطوير المفروضة والموارد المتاحة **(المادية والبشرية)**.

- يدرك مهندس البرمجيات انه يجب عليه العمل من خلال القيود التنظيمية والمالية. لذلك عليه ان يبحث عن الحلول ضمن هذه القيود.

# مهندس البرمجيات ( Software Engineer)

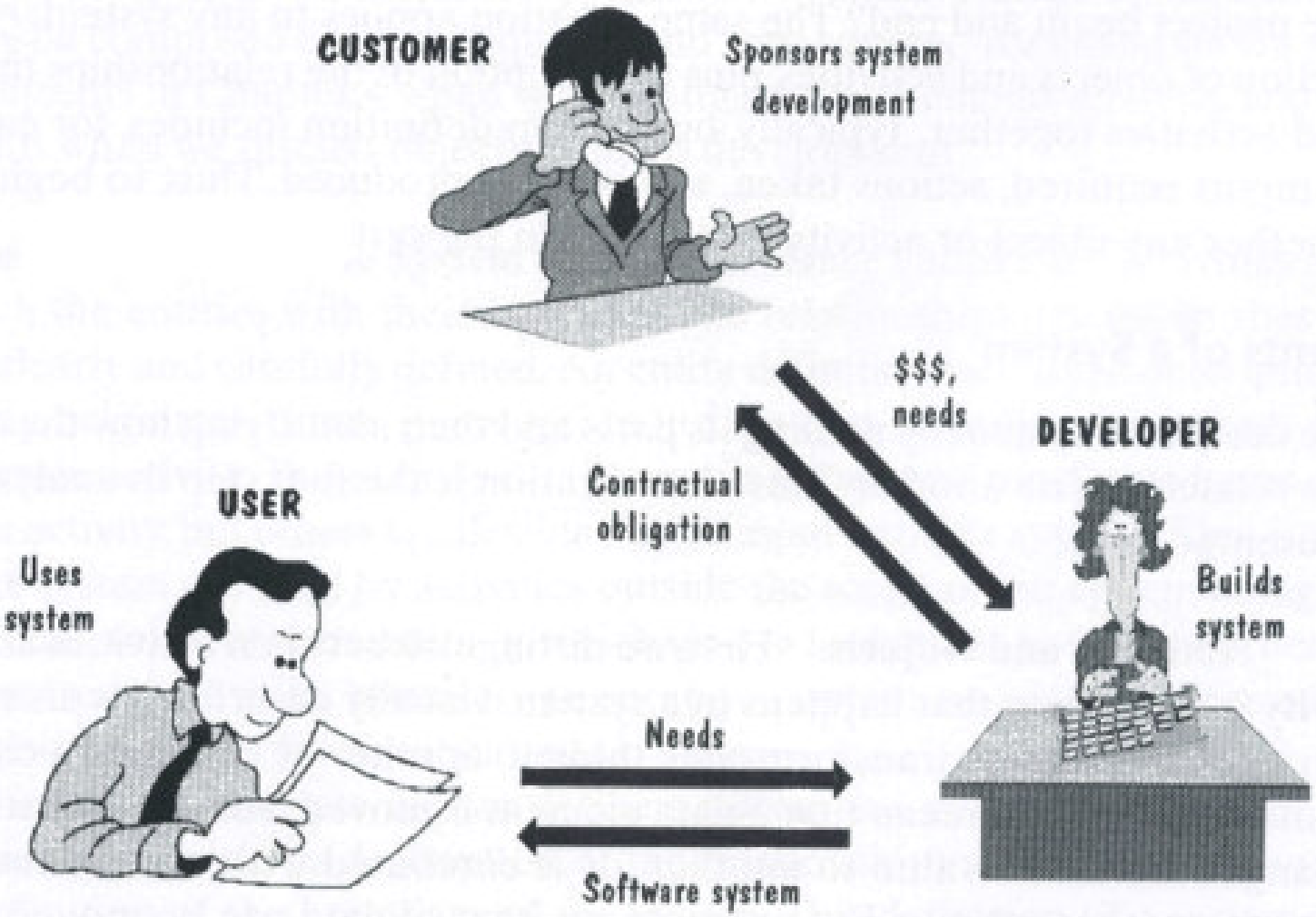
- لذلك فان هندسة البرمجيات تهتم بتصميم وتطوير برامج ذات جودة عالية. السؤال التي يجب سؤاله هنا: من يشارك في عملية التصميم وتطوير البرامج؟؟

- المشاركون في عملية صناعة البرنامج، عادة ما يندرجون تحت ثلاث مجموعات:

١. **الزبون (Customer)**: وهو الشركة (او الشخص) الممولة لمشروع تطوير البرنامج المطلوب.
٢. **المستخدم (User)**: الشخص ( او مجموعة الاشخاص)الذي سوف يقوم فعلا باستعمال البرنامج والتعامل معه مباشرة.

٣. **المطور ( Developer)**: وهو الشركة (او الشخص) الذي سوف يقوم بتطوير البرنامج لصالح الزبون. الشكل التالي يبين العلاقة بين الاجزاء المذكورة انفا:

# مهندس البرمجيات ( Software Engineer)



# مهندس البرمجيات ( Software Engineer)

- **سؤال : ما هي الاسئلة التي يجب على مهندس البرامجيات ان يسأل نفسه عند تصميم هذا للنظام ؟**

١. تغير العمليات Process Change : أي هل النظام المقترح سيؤدي الى تغير في شكل معالجة العمليات فمثلا العمليات الحسابية في احد الشركات فهل نظامنا سوف يؤدي الى تحول تلك العمليات الى عمليات الية داخل الحاسوب ام لا ؟

٢. تغير وظيفي Job Change: أي هل طبيعة العمل للموظفين ستتغير بمعنى اخر هل سيبقى الموظفون في وظائفهم بحيث سيحتاجون الى اعادة تاهيل ام لا؟

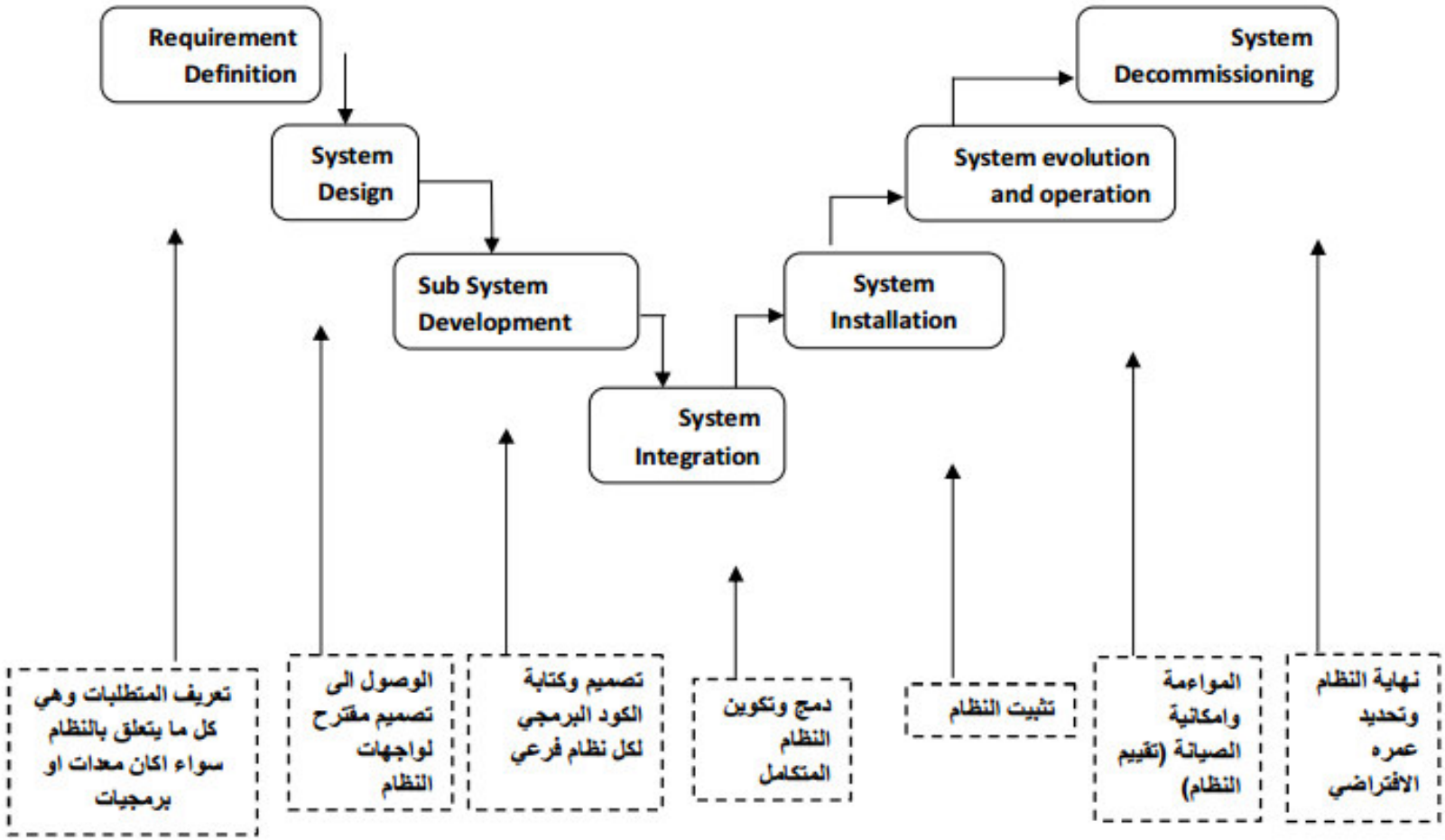
# مهندس البرمجيات ( Software Engineer)

٣. تغير المنظمة Organization Change: هل هذا النظام سيؤدي الى مايسمى بالتغيرات المنظمية أي هل ستتغير هيكلية المنظمة ام لا فعلى سبيل المثال ادارة الارشفة فهي مستقبلا ايلة الى الاختفاء مع ظهور انظمة قواعد البيانات ذات الامنية والكفائة العالية .

- وبعد ان يسأل مهندس البرامجيات نفسه الاسئلة السابقة ينتقل الى "خطوات هندسة البرامجيات".

# خطوات هندسة البرمجيات Software Engineering Process

- ويمكن توضيح تلك الخطوات في الشكل التالي:



# ما الفرق بين هندسة البرمجيات وهندسة النظم

**هندسة النظم** : تهتم هندسة النظم بجميع الجوانب المتعلقة بتطوير النظم المعتمدة على الحاسوب بما فيها من تصميم وبناء للتجهيزات والبرمجيات ومن عناية بإجرائية التطوير وتهتم بجميع نواحي الكمبيوتر من معدات وبرمجيات وعمليات هندسيه

**أما هندسة البرمجيات** فهي ذلك الجزء من هذه الإجرائية الذي يهتم بتطوير البنية البرمجية الأساسية والتحكم والتطبيقات وبناء المعطيات التي تدخل في تكوين هذا النظام وتعتبر هندسة البرمجيات هي جزء من الأنظمة الهندسية.

- الأنظمة الهندسية تهتم بالمواصفات ، التخطيط، التكامل والتطبيق للأنظمة.
- وتعتبر هندسة النظم أقدم من هندسة البرمجيات فقد تمكن المهندسون وخلال الأعوام المائة الماضية من بناء وتجميع العديد من الأنظمة الضخمة كالطائرات والمعامل.

# هندسة البرمجيات

## المحاضرة الثانية

## تكنولوجيا الطبقات، نماذج عمليات البرمجيات

مدرس المادة: م.م نور حسن حسون

# هندسة البرمجيات – تكنولوجيا طبقية:

- تعريف Fritz Bauer الذي وضعه لهندسة البرمجيات ينص علي التالي:

- ***(هندسة البرمجيات) هي استخدام مبادئ هندسية صحيحة للحصول اقتصاديا علي برمجيات موثوقة وتعمل بكفاءة علي كمبيوترات حقيقية.***

- هذا التعريف لا يقول الكثير عن المظاهر التقنية لجودة البرمجيات.
- لا يذكر مباشرة الحاجة الي ارضاء الزبون او تسليم المنتج في الوقت المناسب
- لم يذكر اهمية عملية البرمجة الناضجة

# هندسة البرمجيات – تكنولوجيا طبقية:

- وضع معهد المهندسين الكهربائيين والالكترونيين IEEE تعريفا اشمل وهو:

- ***(هندسة البرمجيات )هي تطبيق منهج مرتب ومنظم وقابل للقياس لعمليات تطوير وتشغيل وصيانة البرمجيات اي تطبيق الهندسة علي البرمحيات.***

- لذلك فان هندسة البرمجيات هي تكنولوجيا طبقية (Layered Technology) . ان حجر الاساس الذي يدعم هندسة البرمجيات هو التركيز على:

١. الجودة (Quality): وهي من اهم اهداف استخدام هندسة البرمجيات في عملية بناء وتطوير البرمجيات حيث تأخذ بعين الاعتبار تخصيصات المستخدم، ومتطلباته على جميع المستويات.

# هندسة البرمجيات – تكنولوجيا طبقية:

- **المعالجة( Process)**: هي عملية اختيار نوع المعالجة المستخدمة في بناء البرنامج وهنا يبدا مدير العمل (Software Developer) بتحديد نوعية وطريقة المعالجة المطلوبة مثلا هل يستخدم طريقة الخطية وتدفق الشلال ( Waterfall) Approachاو طريقة ( Exploratory Approach)او اي طريقة اخرى في العمل ؟ مع مراعاة تلائم نوعية المعالجة مع طبيعة النظام المراد بناؤه .

- **الطرق( Methods)** : هنا مصمم النظام يبدا باخيار طريقة العمل مثلا هل يريد استخدام ( Object _Oriented Methods) او طرق الكلاسيكية بالعمل وهذة الطبقة توفر تقنية الاجابة عن "How" كيف نبني البرانامج

# هندسة البرمجيات – تكنولوجيا طبقية:

- **الادوات(Tools) :** وهذه الطبقة توفر ادوات اتوماتيكية او شبة مؤتمتة في توفير ادوت وبرمجيات تساعد المصمم في عمله ومثال عنها الـ (Computer Aided Software Engineering (CASE))والمخطط التالي يبين طبقات هندسة البرمجيات.

مخطط (1) يوضح طبقات هندسة برامجيات

# نماذج عمليات البرمجيات Software Process Models

- النموذج عبارة عن تمثيل مبسط لدورة حياة تطوير النظام حيث تعرض هذه العمليات من منظور خاص. من امثلة منظور العمليات المستخدمة: منظور تتابع العمل وتتابع الاطوار، ومنظور تدفق العمليات وتدفق البيانات (تدفق المعلومات)، ومنظور قواعد واعمال( تحديد اعمال).

- والنماذج بطبيعتها هي تبسيط لدورة حياة النظام عبارة عن موجز مجرد للعمليات الفعلية الموصوفة، وقد يحتوي على الاطوار التي هي جزء من عمليات البرمجيات التي ينشغل بها العاملون في هندسة البرمجيات.

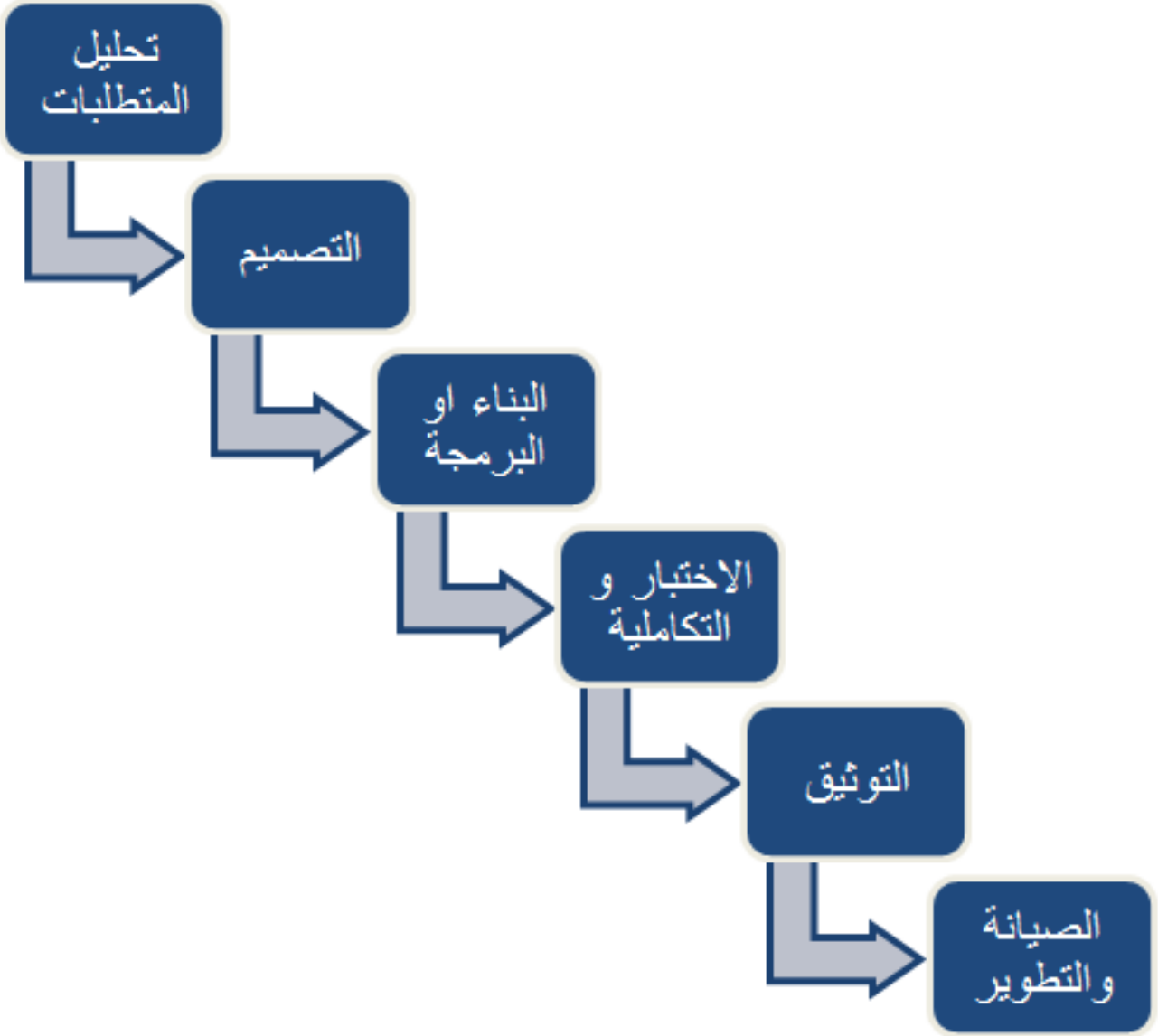
# نماذج عمليات البرمجيات Software Process Models

## ١. النموذج الانحداري او الشلالي Waterfall Model

- في هذا النموذج تسير دورة الحياة بشكل تدريجي بدأ من الخطوة (١) وحتى الخطوة (٨)، وكما يظهر بالشكل (١) فإن كل مرحلة تبدأ بعد الانتهاء من المرحلة التي تسبقها مباشرة.

- يتميز النموذج الانحداري بالبساطة، ولذا فإنه يسّهل على المطور توضيح كيفية سير العمل بالمشروع للعميل (الذي عادة لا يعرف الكثير عن صنع البرمجيات) والمراحل المتبقية من العمل. وقد كان هذا النموذج أساس عمل كثير من المؤسسات لفترة طويلة مثل وزارة الدفاع الامريكية، واستنبط منه العديد من النماذج الاكثر تعقيدا. الشكل التالي يبن مراحل تطوير البرمجيات في النظام الانحداري

# نماذج عمليات البرمجيات Software Process Models



# نماذج عمليات البرمجيات Software Process Models

- **مراحل النموذج**

١. **المرحلة الأولى : تحليل المتطلبات (Requirements Analysis)**

- في هذه المرحلة يقوم محلل النظام او مهندس البرمجيات بتحديد متطلبات النظام من برمجيات، ومعدات ،و المهام والوظائف التي سيقوم بها البرنامج ، وصف هذه المهام بدقة تامّة ،كما يتم عمل دراسة الجدوى لهذا البرنامج

- ، فالعميل في هذه المرحلة يضع تصوراً للبرنامج ليقوم بعمليات معينة وهنا تأتي مهمة مهندس البرمجيات في استخلاص هذه الأفكار الخاصة بالعميل وتحديدها ، لذلك فهي تتطلب مهارة عالية في التعامل مع العملاء والقدرة على التحليل الصحيح. ينتج في نهاية هذه المرحلة وثيقة تدعى جدول الشروط والمواصفات.

- حسب النموذج الإنحداري فإنه يجب على المطورين إنهاء مرحلة تحليل النظام بشكل تام قبل البدء في التصميم، هذه المرحلة قد تتطلب وقت طويل في بعض المشاريع وقد تمر عدة سنوات قبل أن يرى البرنامج النهائي النور.

# نماذج عمليات البرمجيات Software Process Models

١. **المرحلة الثانية : التصميم (Design)**

- عملية التصميم العام للنظام هي عبارة ترتيب النوافذ المختلفة للنظام وجعلها تعمل بطريقة متكاملة لتحقيق الأهداف الخاصة بالنظام ، ويجب دراسة وتقويم مجموعة من العناصر الهامة والمؤثرة في عملية التصميم والتي سوف نتعرف عليها بشكل أكثر تفصيلاً في المقالات التالية.

٣. **المرحلة الثالثة : البناء او البرمجة (Implementation and coding)**

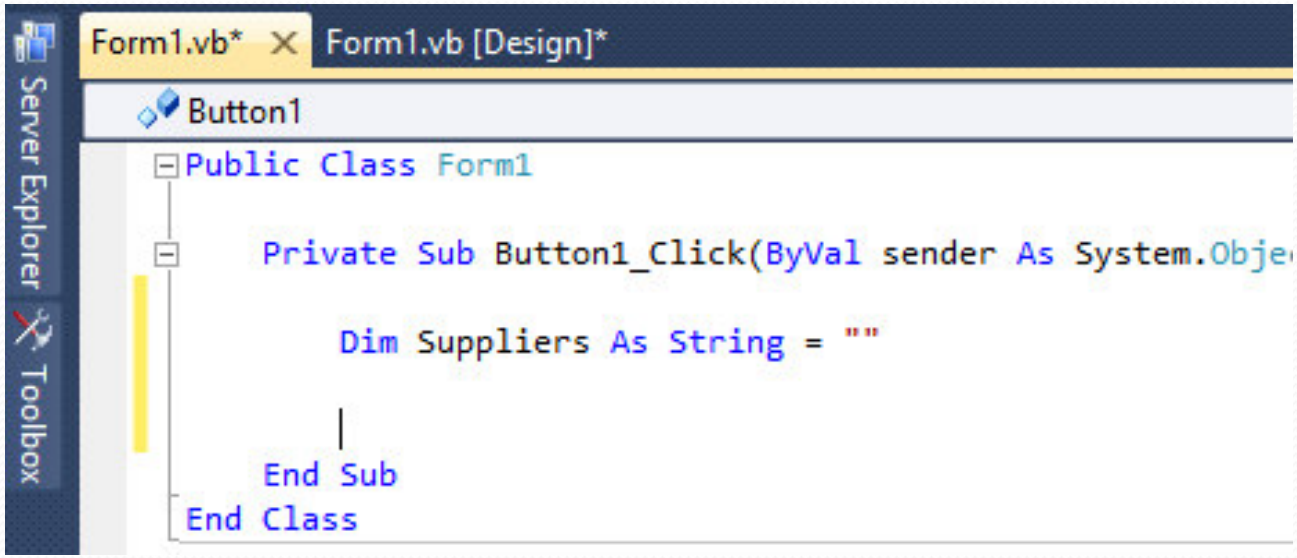
- مرحلة كتابة الأكواد البرمجية تعد جزءاُ من التصميم ، حيث يجب عليك كمبرمج تمثيل الأكواد البرمجية في مخيلتك أثناء تصميم النوافذ ، كما يجب وضع الموصفات القياسية لكتابة الأكواد البرمجية في عين الاعتبار ،مثل التسميات للمتغيرات والثوابت بشكل واضح وبدون اختصار فإن كنت مثلا ستصرح عن متغير خاص بالموردين فلا تقوم بالتصريح عنه كالتالي – باستخدام فيجوال بيسك دوت نت :-

# نماذج عمليات البرمجيات Software Process Models

```
Form1.vb*   Form1.vb [Design]*
Button1
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Obje

        Dim s As String

    End Sub
End Class
```

- فتخيل مثلا أن هناك مبرمج آخر سوف يقوم بالقراءة والتعديل على الأكواد فماذا سيفهم من متغير باسم (S) ؟.لذلك وجب عليك التصريح عن المتغير بشكل يوحي بمضمون عمل هذا المتغير كالتالي – باستخدام فيجوال بيسك دوت نت :

# نماذج عمليات البرمجيات Software Process Models




٤. **المرحلة الرابعة : الاختبار و التكاملية (Testing and Integrative)**

- يتم جمع الكتل البرمجية مع بعضها و اختبار النظام البرمجي للتأكد من موافقته للشروط و المواصفات المتفق عليه عند تحليل النظام ، و خاصة اذا كانت الكتل البرمجية قد صممت وكتبت أكوادها من قبل عدة أعضاء في فريق تطوير النظام .

# نماذج عمليات البرمجيات Software Process Models

٥. المرحلة الخامسة : التوثيق (Documentation)

- وهي مرحلة هامة من مراحل بناء النظام البرمجي حيث يتم توثيق البناء الداخلي للبرنامج وذلك بغرض الصيانة والتطوير، ويفضل عادة أن يترافق التوثيق مع كل مرحلة من المراحل السابقة واللاحقة، وأن يكون هناك فريق خاص يهتم بعملية التوثيق لجميع المشاكل والحلول التي يمكن أن تظهر أثناء بناء النظام البرمجي

- وبدون التوثيق قد يصل مطور النظام البرمجي إلى مرحلة لا يعود بعدها قادراً على متابعة صيانة النظام وتطويره مما يزيد من الكلفة المادية والزمنية الخاصة بهذه البرمجية إلى حدود غير متوقعة، أو بمعنى آخر الفشل في بناء نظام برمجي ذات جودة عالية .

# نماذج عمليات البرمجيات Software Process Models

٦. **المرحلة السادسة : الصيانة و التطوير (Maintenance and development)**

- إن هذه المرحلة هي المرحلة الأطول في حياة النظام البرمجي لبقاء النظام قادراً على مواكبة التطورات و المعدات الحديثة، جزء من هذه المرحلة يكون في تصحيح الأخطاء البرمجية التي تظهر من كثرة الاستخدام وإدخال الكثير من البيانات والجزء الآخر يكون في التطوير و إضافة إمكانيات جديدة.

# نماذج عمليات البرمجيات Software Process Models

- **ايجابيات النموذج (Model Advantages)**

١. نموذج سهل للفهم
٢. سهل للإدارة
٣. المراحل تكتمل وتعالج مرحلة تلو الاخرى
٤. العمل يقسم الى مشاريع صغيرة حيث المتطلبات تصبح سهلة للفهم
٥. يفضل في المشاريع حيث الجودة هي اكثر اهمية بالمقارنة مع الجدول الزمني والتكلفة

# نماذج عمليات البرمجيات Software Process Models

- **السلبيات النموذج (Model Disadvantages)**

١. لا يمكن ان تعود خطوة ، اذا مرحلة ما يوجد بها خطأ لا يمكن الرجوع اليها وتعديلها لا نها تصبح العمليات معقدة في المرحلة، حتى لو تغيير بسيط في اي مرحلة سابقة يمكن ان يسبب مشاكل كبيرة للمراحل اللاحقة حيث ان جميع المراحل تعتمد على بعضها.
٢. نسبة كبيرة من المخاطر واحتمال حصول اخطاء
٣. ليس نموذج جيد للمشاريع المعقدة
٤. نموذج ضعيف للمشاريع الطويلة والمستمرة
٥. غير مناسب للمشاريع حيث المتطلبات فيها مخاطر عالية من التغيير

# هندسة البرمجيات

## المحاضرة الثانية

## تكنولوجيا الطبقات، نماذج عمليات البرمجيات

مدرس المادة: م.م نور حسن حسون

# هندسة البرمجيات – تكنولوجيا طبقية:

- تعريف Fritz Bauer الذي وضعه لهندسة البرمجيات ينص علي التالي:

- ***(هندسة البرمجيات) هي استخدام مبادئ هندسية صحيحة للحصول اقتصاديا علي برمجيات موثوقة وتعمل بكفاءة علي كمبيوترات حقيقية.***

- هذا التعريف لا يقول الكثير عن المظاهر التقنية لجودة البرمجيات.
- لا يذكر مباشرة الحاجة الي ارضاء الزبون او تسليم المنتج في الوقت المناسب
- لم يذكر اهمية عملية البرمجة الناضجة

# هندسة البرمجيات – تكنولوجيا طبقية:

- وضع معهد المهندسين الكهربائيين والالكترونيين IEEE تعريفا اشمل وهو:

- ***(هندسة البرمجيات )هي تطبيق منهج مرتب ومنظم وقابل للقياس لعمليات تطوير وتشغيل وصيانة البرمجيات اي تطبيق الهندسة علي البرمحيات.***

- لذلك فان هندسة البرمجيات هي تكنولوجيا طبقية (Layered Technology) . ان حجر الاساس الذي يدعم هندسة البرمجيات هو التركيز على:

١. الجودة (Quality): وهي من اهم اهداف استخدام هندسة البرمجيات في عملية بناء وتطوير البرمجيات حيث تأخذ بعين الاعتبار تخصيصات المستخدم، ومتطلباته على جميع المستويات.

# هندسة البرمجيات – تكنولوجيا طبقية:

- **المعالجة( Process)**: هي عملية اختيار نوع المعالجة المستخدمة في بناء البرنامج وهنا يبدا مدير العمل (Software Developer) بتحديد نوعية وطريقة المعالجة المطلوبة مثلا هل يستخدم طريقة الخطية وتدفق الشلال ( Waterfall) Approachاو طريقة ( Exploratory Approach)او اي طريقة اخرى في العمل ؟ مع مراعاة تلائم نوعية المعالجة مع طبيعة النظام المراد بناؤه .

- **الطرق( Methods)** : هنا مصمم النظام يبدا باخيار طريقة العمل مثلا هل يريد استخدام ( Object _Oriented Methods) او طرق الكلاسيكية بالعمل وهذة الطبقة توفر تقنية الاجابة عن "How" كيف نبني البرانامج

# هندسة البرمجيات – تكنولوجيا طبقية:

- **الادوات(Tools) :** وهذه الطبقة توفر ادوات اتوماتيكية او شبة مؤتمتة في توفير ادوت وبرمجيات تساعد المصمم في عمله ومثال عنها الـ (Computer Aided Software Engineering (CASE))والمخطط التالي يبين طبقات هندسة البرمجيات.

مخطط (1) يوضح طبقات هندسة برامجيات

# نماذج عمليات البرمجيات Software Process Models

- النموذج عبارة عن تمثيل مبسط لدورة حياة تطوير النظام حيث تعرض هذه العمليات من منظور خاص. من امثلة منظور العمليات المستخدمة: منظور تتابع العمل وتتابع الاطوار، ومنظور تدفق العمليات وتدفق البيانات (تدفق المعلومات)، ومنظور قواعد واعمال( تحديد اعمال).

- والنماذج بطبيعتها هي تبسيط لدورة حياة النظام عبارة عن موجز مجرد للعمليات الفعلية الموصوفة، وقد يحتوي على الاطوار التي هي جزء من عمليات البرمجيات التي ينشغل بها العاملون في هندسة البرمجيات.

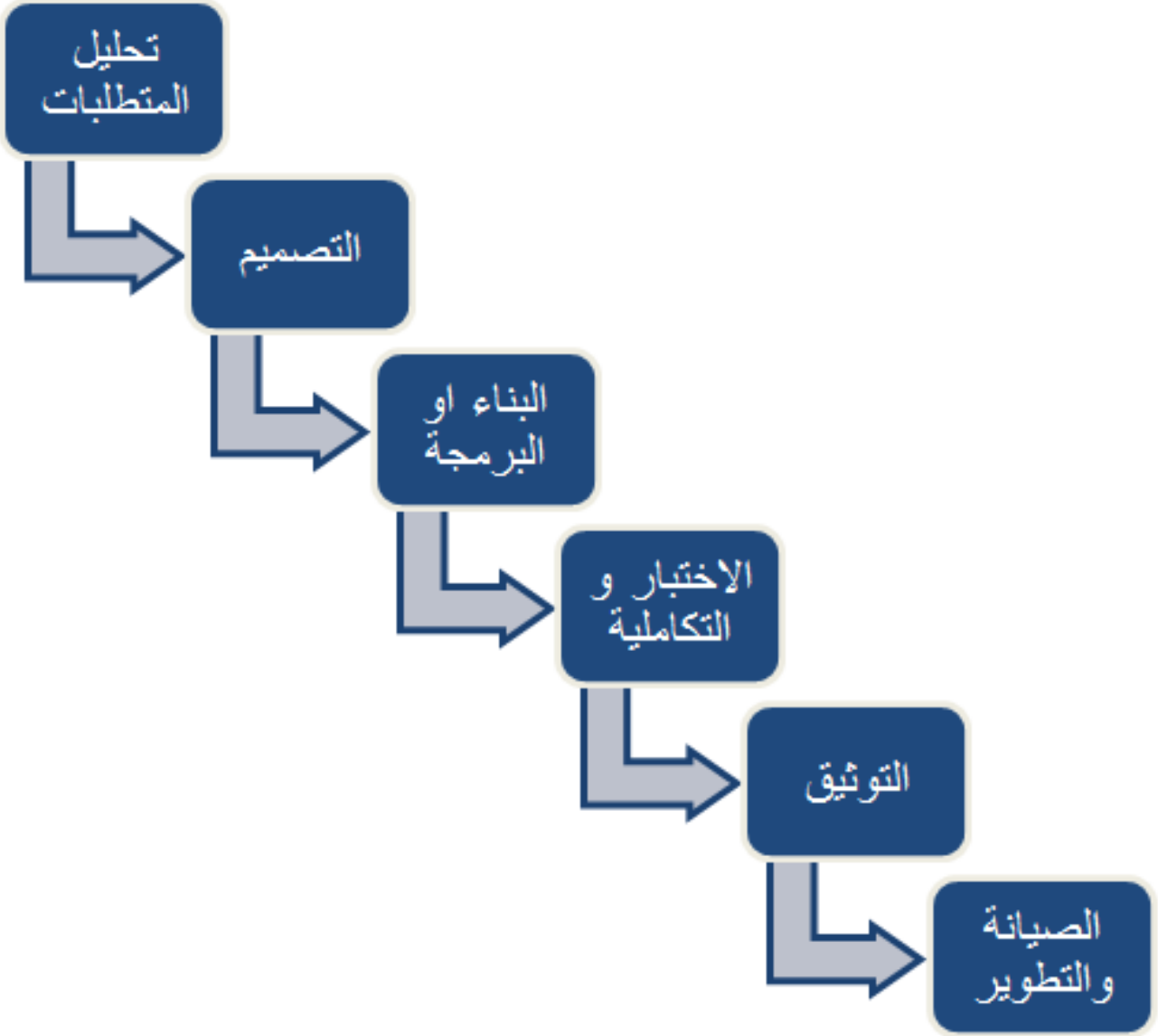
# نماذج عمليات البرمجيات Software Process Models

## ١. النموذج الانحداري او الشلالي Waterfall Model

- في هذا النموذج تسير دورة الحياة بشكل تدريجي بدأ من الخطوة (١) وحتى الخطوة (٨)، وكما يظهر بالشكل (١) فإن كل مرحلة تبدأ بعد الانتهاء من المرحلة التي تسبقها مباشرة.

- يتميز النموذج الانحداري بالبساطة، ولذا فإنه يسّهل على المطور توضيح كيفية سير العمل بالمشروع للعميل (الذي عادة لا يعرف الكثير عن صنع البرمجيات) والمراحل المتبقية من العمل. وقد كان هذا النموذج أساس عمل كثير من المؤسسات لفترة طويلة مثل وزارة الدفاع الامريكية، واستنبط منه العديد من النماذج الاكثر تعقيدا. الشكل التالي يبن مراحل تطوير البرمجيات في النظام الانحداري

# نماذج عمليات البرمجيات Software Process Models



# نماذج عمليات البرمجيات Software Process Models

- **مراحل النموذج**

١. **المرحلة الأولى : تحليل المتطلبات (Requirements Analysis)**

- في هذه المرحلة يقوم محلل النظام او مهندس البرمجيات بتحديد متطلبات النظام من برمجيات، ومعدات ،و المهام والوظائف التي سيقوم بها البرنامج ، وصف هذه المهام بدقة تامّة ،كما يتم عمل دراسة الجدوى لهذا البرنامج

- ، فالعميل في هذه المرحلة يضع تصوراً للبرنامج ليقوم بعمليات معينة وهنا تأتي مهمة مهندس البرمجيات في استخلاص هذه الأفكار الخاصة بالعميل وتحديدها ، لذلك فهي تتطلب مهارة عالية في التعامل مع العملاء والقدرة على التحليل الصحيح. ينتج في نهاية هذه المرحلة وثيقة تدعى جدول الشروط والمواصفات.

- حسب النموذج الإنحداري فإنه يجب على المطورين إنهاء مرحلة تحليل النظام بشكل تام قبل البدء في التصميم، هذه المرحلة قد تتطلب وقت طويل في بعض المشاريع وقد تمر عدة سنوات قبل أن يرى البرنامج النهائي النور.

# نماذج عمليات البرمجيات Software Process Models

١. **المرحلة الثانية : التصميم (Design)**

- عملية التصميم العام للنظام هي عبارة ترتيب النوافذ المختلفة للنظام وجعلها تعمل بطريقة متكاملة لتحقيق الأهداف الخاصة بالنظام ، ويجب دراسة وتقويم مجموعة من العناصر الهامة والمؤثرة في عملية التصميم والتي سوف نتعرف عليها بشكل أكثر تفصيلاً في المقالات التالية.

٣. **المرحلة الثالثة : البناء او البرمجة (Implementation and coding)**

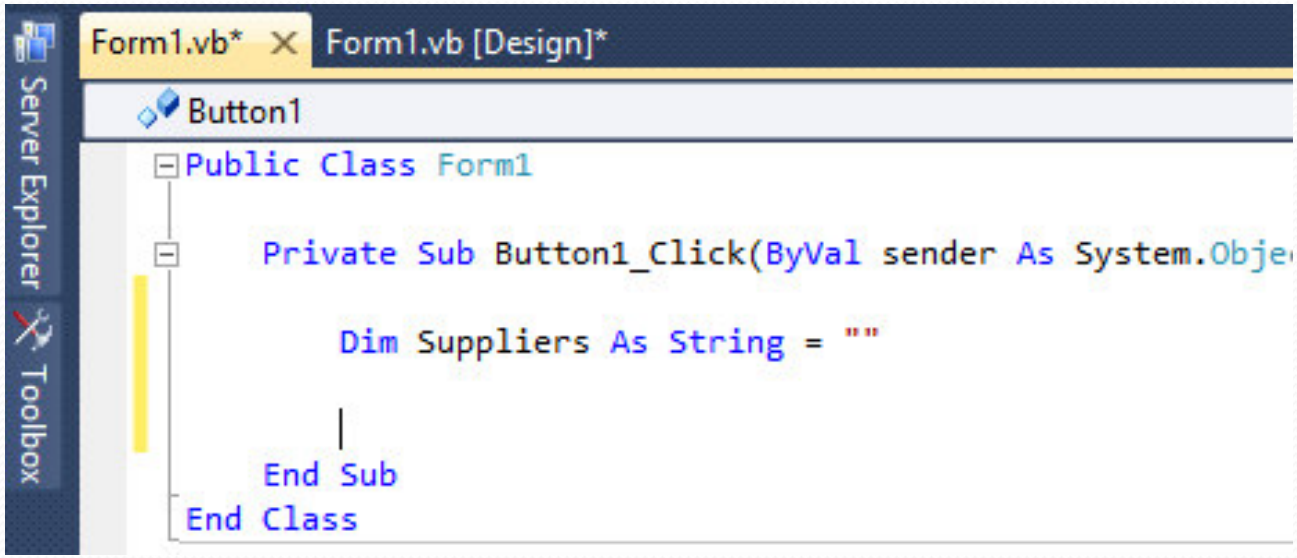
- مرحلة كتابة الأكواد البرمجية تعد جزءاُ من التصميم ، حيث يجب عليك كمبرمج تمثيل الأكواد البرمجية في مخيلتك أثناء تصميم النوافذ ، كما يجب وضع الموصفات القياسية لكتابة الأكواد البرمجية في عين الاعتبار ،مثل التسميات للمتغيرات والثوابت بشكل واضح وبدون اختصار فإن كنت مثلا ستصرح عن متغير خاص بالموردين فلا تقوم بالتصريح عنه كالتالي – باستخدام فيجوال بيسك دوت نت :-

# نماذج عمليات البرمجيات Software Process Models

```
Form1.vb*   Form1.vb [Design]*
Button1
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Obje

        Dim s As String

    End Sub
End Class
```

- فتخيل مثلا أن هناك مبرمج آخر سوف يقوم بالقراءة والتعديل على الأكواد فماذا سيفهم من متغير باسم (S) ؟.لذلك وجب عليك التصريح عن المتغير بشكل يوحي بمضمون عمل هذا المتغير كالتالي – باستخدام فيجوال بيسك دوت نت :

# نماذج عمليات البرمجيات Software Process Models




٤. **المرحلة الرابعة : الاختبار و التكاملية (Testing and Integrative)**

- يتم جمع الكتل البرمجية مع بعضها و اختبار النظام البرمجي للتأكد من موافقته للشروط و المواصفات المتفق عليه عند تحليل النظام ، و خاصة اذا كانت الكتل البرمجية قد صممت وكتبت أكوادها من قبل عدة أعضاء في فريق تطوير النظام .

# نماذج عمليات البرمجيات Software Process Models

٥. المرحلة الخامسة : التوثيق (Documentation)

- وهي مرحلة هامة من مراحل بناء النظام البرمجي حيث يتم توثيق البناء الداخلي للبرنامج وذلك بغرض الصيانة والتطوير، ويفضل عادة أن يترافق التوثيق مع كل مرحلة من المراحل السابقة واللاحقة، وأن يكون هناك فريق خاص يهتم بعملية التوثيق لجميع المشاكل والحلول التي يمكن أن تظهر أثناء بناء النظام البرمجي

- وبدون التوثيق قد يصل مطور النظام البرمجي إلى مرحلة لا يعود بعدها قادراً على متابعة صيانة النظام وتطويره مما يزيد من الكلفة المادية والزمنية الخاصة بهذه البرمجية إلى حدود غير متوقعة، أو بمعنى آخر الفشل في بناء نظام برمجي ذات جودة عالية .

# نماذج عمليات البرمجيات Software Process Models

٦. **المرحلة السادسة : الصيانة و التطوير (Maintenance and development)**

- إن هذه المرحلة هي المرحلة الأطول في حياة النظام البرمجي لبقاء النظام قادراً على مواكبة التطورات و المعدات الحديثة، جزء من هذه المرحلة يكون في تصحيح الأخطاء البرمجية التي تظهر من كثرة الاستخدام وإدخال الكثير من البيانات والجزء الآخر يكون في التطوير و إضافة إمكانيات جديدة.

# نماذج عمليات البرمجيات Software Process Models

- **ايجابيات النموذج (Model Advantages)**

١. نموذج سهل للفهم
٢. سهل للإدارة
٣. المراحل تكتمل وتعالج مرحلة تلو الاخرى
٤. العمل يقسم الى مشاريع صغيرة حيث المتطلبات تصبح سهلة للفهم
٥. يفضل في المشاريع حيث الجودة هي اكثر اهمية بالمقارنة مع الجدول الزمني والتكلفة

# نماذج عمليات البرمجيات Software Process Models

- **السلبيات النموذج (Model Disadvantages)**

١. لا يمكن ان تعود خطوة ، اذا مرحلة ما يوجد بها خطأ لا يمكن الرجوع اليها وتعديلها لا نها تصبح العمليات معقدة في المرحلة، حتى لو تغيير بسيط في اي مرحلة سابقة يمكن ان يسبب مشاكل كبيرة للمراحل اللاحقة حيث ان جميع المراحل تعتمد على بعضها.
٢. نسبة كبيرة من المخاطر واحتمال حصول اخطاء
٣. ليس نموذج جيد للمشاريع المعقدة
٤. نموذج ضعيف للمشاريع الطويلة والمستمرة
٥. غير مناسب للمشاريع حيث المتطلبات فيها مخاطر عالية من التغيير

# هندسة البرمجيات

## المحاضرة الثالثة

## نماذج عمليات البرمجيات

مدرس المادة: م.م نور حسن حسون

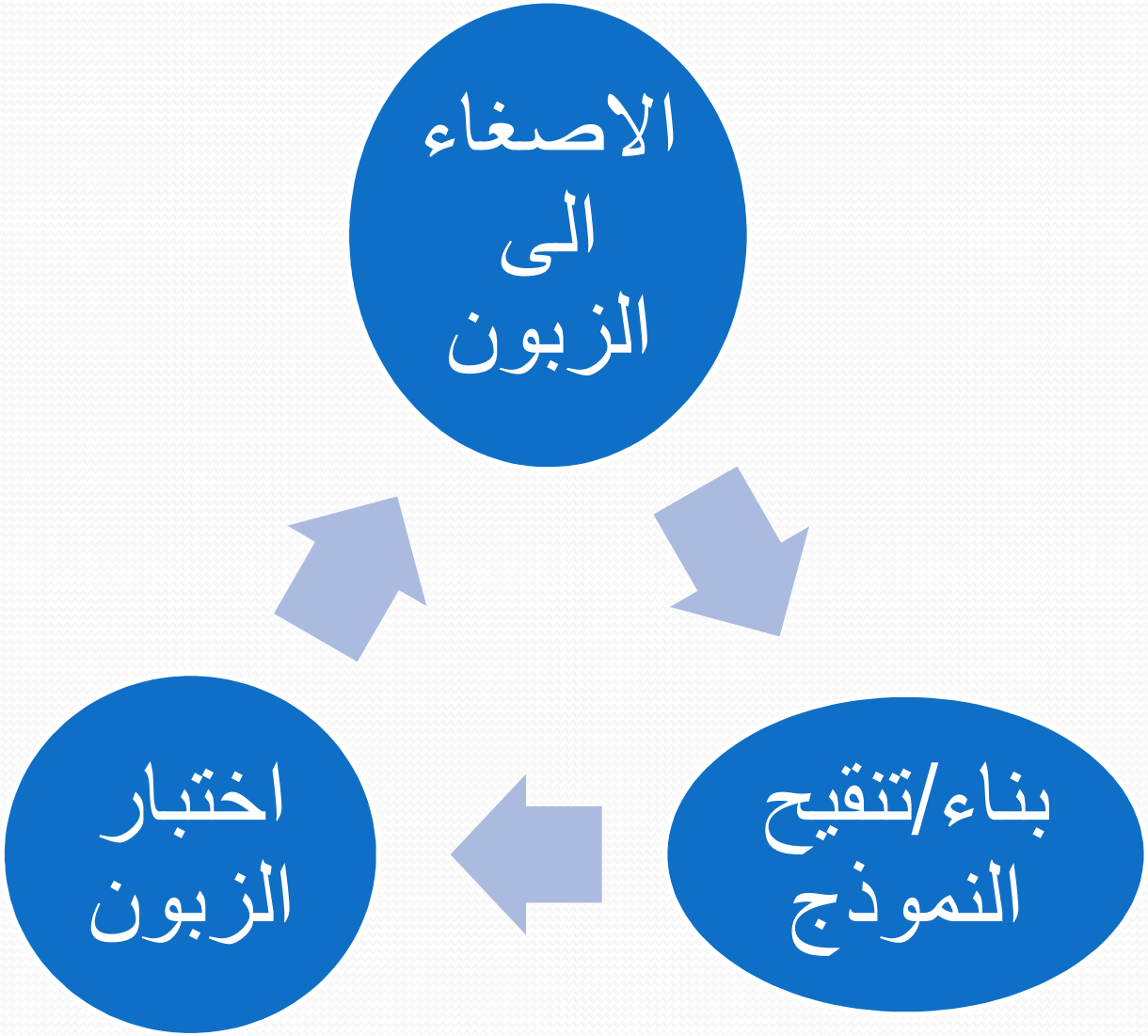
# نموذج "النمذجة الأولية" Prototype Model

- يبدا هذا النموذج من خلال تعريف الزبون مجموعة من الأهداف العامة للبرنامج، ولا يحدد بالتفصيل كل متطلبات الادخال أو المعالجة أو الاخراج،

- في بعض الحالات قد يكون مطور غير متأكد من فعالية الخوارزمية او النموذج، أو الشكل الذي يجب أن يأخذه تفاعل الإنسان مع الآلة، لذلك فان نموذج النمذجة الأولية (Prototyping Paradigm) جاء ليقدم الطريقة الفضلى في حل المشاكل في هذه الحالات وحالات كثيرة غيرها.

- الشكل التالي يبين نموذج النمذجة الاولية:

# نموذج "النمذجة الأولية" Prototype Model



# نموذج "النمذجة الأولية" Prototype Model

- خطوات هذه النموذج تكون كالاتي:

١. الاصغاء الى الزبون Listen to customer

- يبدا هذا النموذج بجمع المتطلبات النظام من خلال اجتماع يحصل بين مطور النظام والزبون لتعريف وتحديد الأهداف الإجمالية للبرنامج.

٢. بناء النموذج build/revise mock-up

- تطوير نموذج أولي (مبدئي ، تجريبي): استنادا إلى حاجات الزبون او المستخدم، حيث يعمل مصممو النظم على تأمين نموذج سريع بواسطة البرمجيات والأدوات المساعدة.

# نموذج "النمذجة الأولية" Prototype Model

٣. اختبار الزبون customer test drives mock-up

- تحديد إذا كان النموذج الأولي مقبولا أم لا (تجربة النموذج).

- في هذه المرحلة يعيد الزبون أو المستخدم تقييم النموذج الأولي، لضمان ان هذا النموذج قد لبى متطلبات تصميم النظام، وكذلك يساع المطور من ايجاد فهم افضل للمتطلبات الواجب تلبيتها للزبون.

- يحدث التكرار في هذه المرحلة لضبط النموذج الاولي ولضمان تحقيق متطلبات الزبون

# نموذج "النمذجة الأولية" Prototype Model

**- مميزات النمذجة الأولية:**

١. يمكن اعتبارها اقتصادية حيث تقل الكلفة بسبب تلافي الأخطاء والالتباس وسوء فهم الاحتياجات في مرحلة مبكرة.

٢. سهلة الفهم حيث تتجه الى بناء النظام خطوة بخطوة، ولا يتم الانتقال الى خطوة تالية إلا بعد التأكد من الخطوة الأولى.

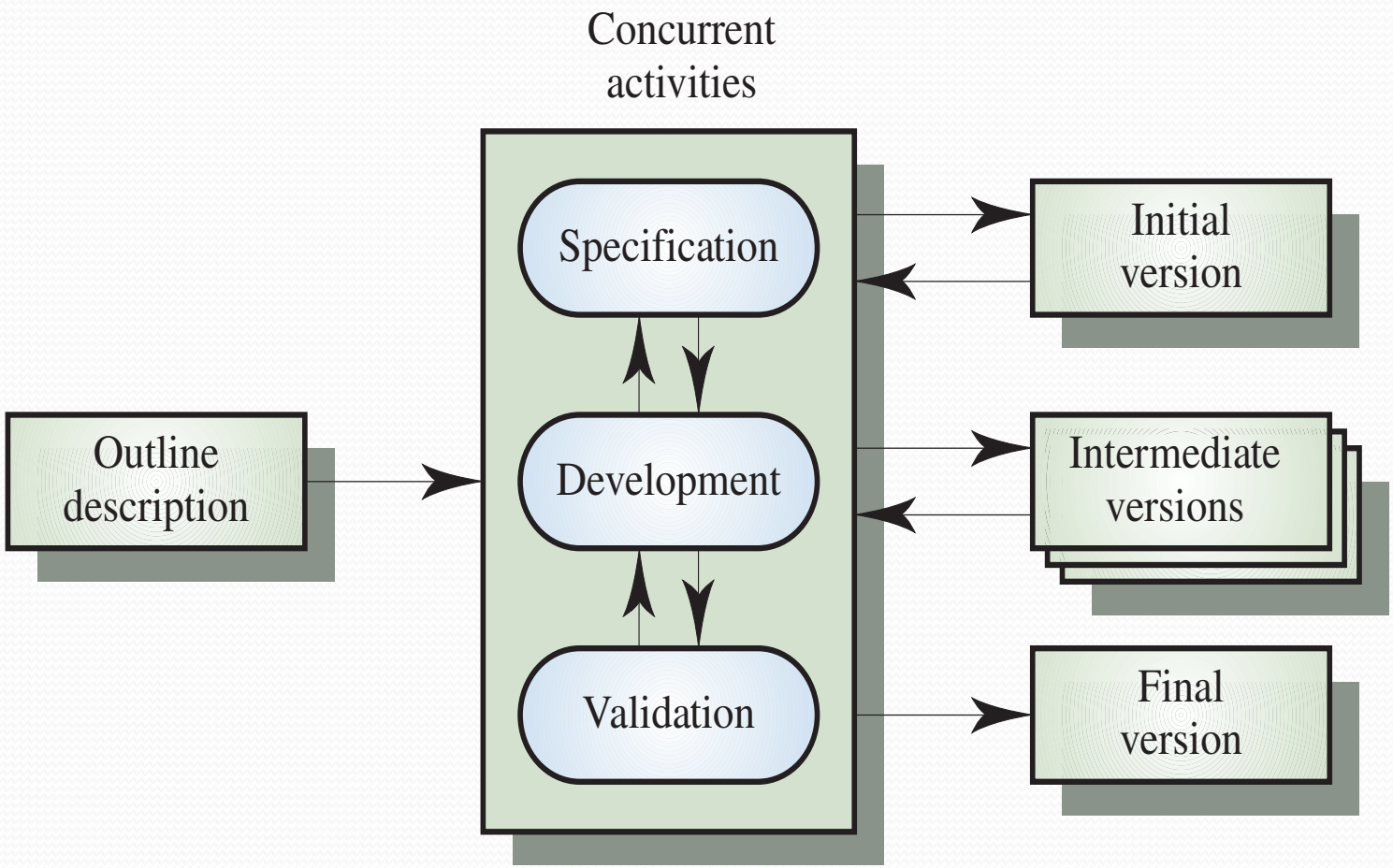
٣. يستوعب المستخدم النظام جيداً قبل تركيبه وتنفيذه بشكل نهائي.

**- عيوب النمذجة الأولية:**

١. معظم المشاريع بالكاد يكون اول نظام يبني قابلا للاستخدام فقد يكون بطئيا جدا او كبيرا جدا او صعب الاستخدام او الثلاثة معا.

٢. اذا لم يتم التحكم بالمراحل يمكن أن تطول عملية النمذجة.

# Prototype Model نموذج "النمذجة الأولية"



# النماذج التطورية Evolutionary Models

- هناك اعترافٌ متنامٍ بأن البرمجيات تتطور على مر الزمن. إذ تتغير متطلبات الزبون والمنتَج مع تَقدم عملية تطوير هذا المنتَج، وهذا ما يجعل المسار المستقيم ( النموذج الانحداري Waterfall Model) باتجاه إنتاجه غير واقعي. لقد صُمِّم النموذج التتابعي الخطي لحالات التطوير المباشر (خط مستقيم). وبمعنى آخر، تفترض هذه الطريقة التتابعية أن النظام كله سيسلَّم بعد اكتمال هذا التتابع الخطي.

- ومن جهة أخرى، فقد صُمِّم نموذج النمذجة الأولية (Prototype Model) لمساعدة الزبون (أو المطوِّر) على فهم المتطلبات، ولم يصمم عموماً لتسليم نظام نهائي. فلَم تُلاحَظ الطبيعة التطورية للبرمجيات في كلا هذين النموذجين التقليديين لهندسة البرمجيات.

# النماذج التطورية Evolutionary Models

- النماذج التطورية evolutionary models هي نماذج تكرارية، وتوصَّف بطريقة تمكِّن مهندس البرمجيات من تطوير نسخ أكثر تعقيداً من البرمجيات، وسنذكر فيما يلي اثنين من هذه النماذج:

١. **النموذج التزايدي (Incremental Model)**

- النموذج المُتزايد: الذي يجمع مكونات من النموذج المتتالي الخطي مطبقة بشكل متكرر، ويقوم النموذج المتزايد بعمل تتابع مخطوط بصيغة متتالية في الترتيب لكن مع تقدم زمن الإنتاج، فينتج كل تتابع خطي برمجيّات متزايدة جاهزة للتسليم.

- هذا النموذج كما يقترحه اسمه يعمل بالتزايد، أي أن التطبيق الناتج لا يُصدر مرة واحدة و إنما على دفعات تسمى بالمتزايدة.

# النماذج التطورية Evolutionary Models

- كمثال، لنأخذ تطبيق على أجهزة المكتب، هذا التطبيق يسمح لك بإضافة عملاء و بياناتهم و بيانات شرائهم كما يسجل كل البضائع في المخزن و يتابع عددها بإنشاء تقارير اسبوعية.

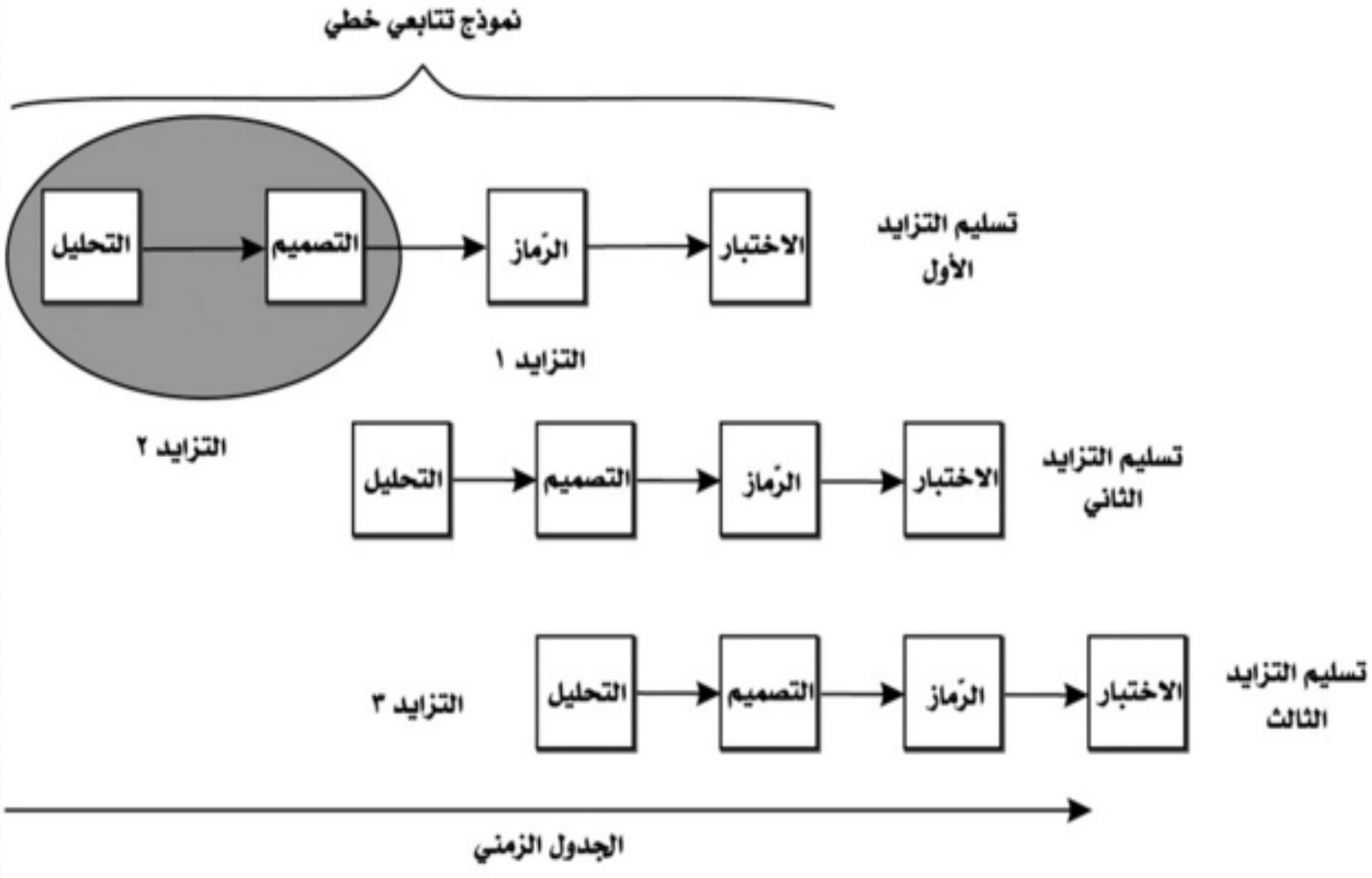
- إذا كان التطبيق سيتم تطويره باستخدام هذا النموذج، فإن أول عملية يقوم بها المطورون هو تقسيم الوظائف إلى مجموعات غير مترابطة، بحيث يتم تطوير كل مجموعة على حدة بشكل متزايدة، يتم البدء بأهم و أعقد الوظائف و كون تحديد وظيفة ما بأنها مهمة راجع لأسباب منها رغبة العميل نفسه ،

# النماذج التطورية Evolutionary Models

- ولنفترض أن إضافة العملاء هي أهم خاصية برأي العميل، ذلك يتم تطبيقها و اختبارها و اطلاقها في النظام و السماح للعميل لا لتجربتها فقط بل و البدء بالاستخدام الفعلي ، حالما تطلق الوظيفة يتم البدء حالاً بتطوير المتزايدة اللاحقة و بنفس المنظومة يتم اختبارها و من ثم دمجها مع الوظيفة السابقة واطلاقها للعميل و هكذا، اذاً نرى أن العميل له دور تفاعلي اساسي في هذا النموذج ، كما أن المطورون يحصلون على رأيه بشكل متتالي و سريع مما يسمح لهم بالتعديل بفاعلية اكبر.

- الشكل يبين النموذج التزايدي.

# النماذج التطورية Evolutionary Models



# النماذج التطورية Evolutionary Models

- **من فوائد هذه النموذج** انه

١. حقق تطورا فظيعا في تطوير البرمجيات
٢. انشئ مفاهيم جديدة، كما انه يتميز بملائمته للعصر الحالي
٣. الحصول على التطبيقات في وقت سريع حيث ان مدة المشاريع تصل الى شهور حينما تمتد مشاريع نموذج الشلال الى سنين .

- ولكن ان من عيوب هذا النموذج انه يطلق نسخ كثيرة من نفس التطبيق، هذا ربما يسبب الانزعاج لدى المستخدم و الذي لا يكون بالضرورة نفس شخص العميل ،

# النماذج التطورية Evolutionary Models

- في مثالنا السابق مثلا يكون العميل هو صاحب الشركة لكنه لا يستخدم النظام فعليا و انما موظفونه، ان الموظفون هم المستخدمون المقيمون لهذا التطبيق و كونهم لا يشترونه منك لا عني ان لا تضعهم في الحسبان ، لأن في النهاية الغرض من التطبيق دائما هو لتحسين و تطوير فاعلية العمل و حين يكون الموظفون منزعجون فإن فاعليتهم ستقل و سينزعج منك العميل في النهاية.

- كن هذا العيب يمكن اصلاحه بسهولة بتقسيم مناسب للوظائف بحيث في كل دفعة يتوجه الاهتمام لقسم معين ، او بالتدريب المناسب لموظفين حال اطلاق نسخة حديدة بحيث لا يشعرون بالضياع ، أو أسوأ: أن يشعروا بالغباء و هو السبب الحقيقي غالباً وراء كل انزعاج من المستخدم .

# النماذج التطورية Evolutionary Models

## ٢. النموذج اللولبي او الحلزوني Spiral Model

- هو نموذج تطوري لعملية البرمجة يقرن الطبيعة التكرارية للنمذجة الأولية بالنواحي النظامية والمحكومة للنموذج التتابعي الخطي
- يقدم إمكانية تطوير سريع لنسخ تزايدية من البرنامج
- يكون الإصدار التزايدي خلال التكرار الأولي نموذجا ورقيا أو نموذجا أوليا
- ينقسم النموذج الحلزوني إلى عدد من نشاطات الهيكل تسمى منطقة المهمة task region هناك عادة ما بين ٣ إلى ٦ مناطق وهي:

# النماذج التطورية Evolutionary Models

١. الاتصال بالزبون Contact the Customer (التواصل الفعال بين الزبون والمطور)،
٢. التخطيط Planning (تعريف الموارد والمسارات الزمنية للمشروع)،
٣. تحليل المخاطرة risk analysis ( قييم المخاطرة التقنية والإدارية)،
٤. الهندسة Engineering (بناء تمثيل أو أكثر للتطبيق)،
٥. البناء والإصدار construction & release (مهمات البناء والاختبار)،
٦. تقويم الزبون customer evaluation (الحصول على تقييم الزبون للعمل).

# النماذج التطورية Evolutionary Models

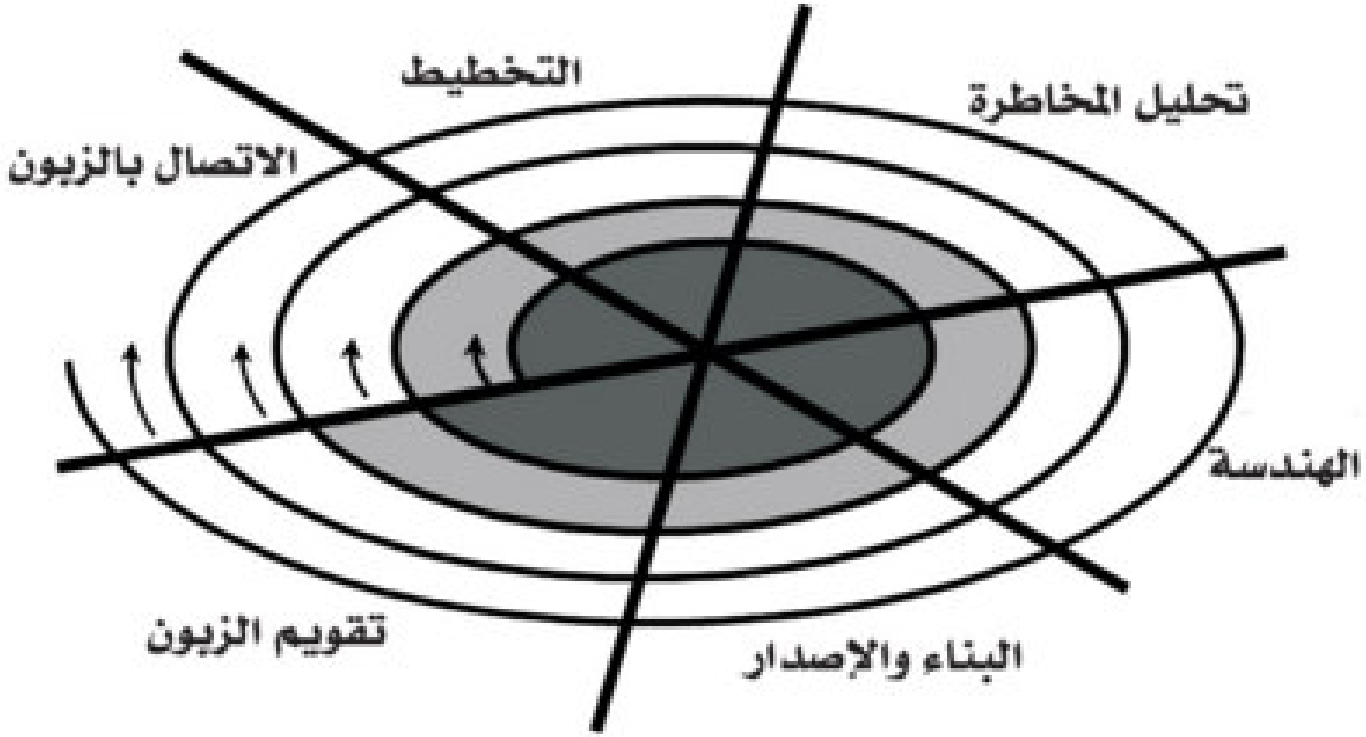
- حالما تبدأ عملية البرمجة التطورية يتحرك فريق هندسة البرمجيات حول الحلزون باتجاه عقارب الساعة بدءا من النواة
- خلافا للنماذج التقليدية التي تنتهي بتسليم البرنامج يمكن تكييف النموذج الحلزوني لتطبيقه على امتداد كامل حياة البرنامج
- يستخدم النموذج الحلزوني النمذجة الأولية كآلية لتقليل المخاطر التقنية ولكنها تمكن المطور من تطبيق النمذجة الأولية في أي مرحلة من مراحل تطور المنتج

**العيوب:**

النموذج الحلزوني "**قلل من المشاكل بشكل كبير**" **ولكن** يمكن أن تظهر نفس المشكله في التي كانت تظهر في الشلال حيث في حال أكتشف المبرمج أن هناك خطأ أو نقص في التصميم لا يمكن العودة للخلف ويجب الانتظار حتى البدء في الدورة الثانية

# النماذج التطورية Evolutionary Models

- الشكل الاتي يبين النموذج اللولبي او الحلزوني



# هندسة البرمجيات

## المحاضرة الرابعة

## نماذج عمليات البرمجيات

مدرس المادة: م.م نور حسن حسون

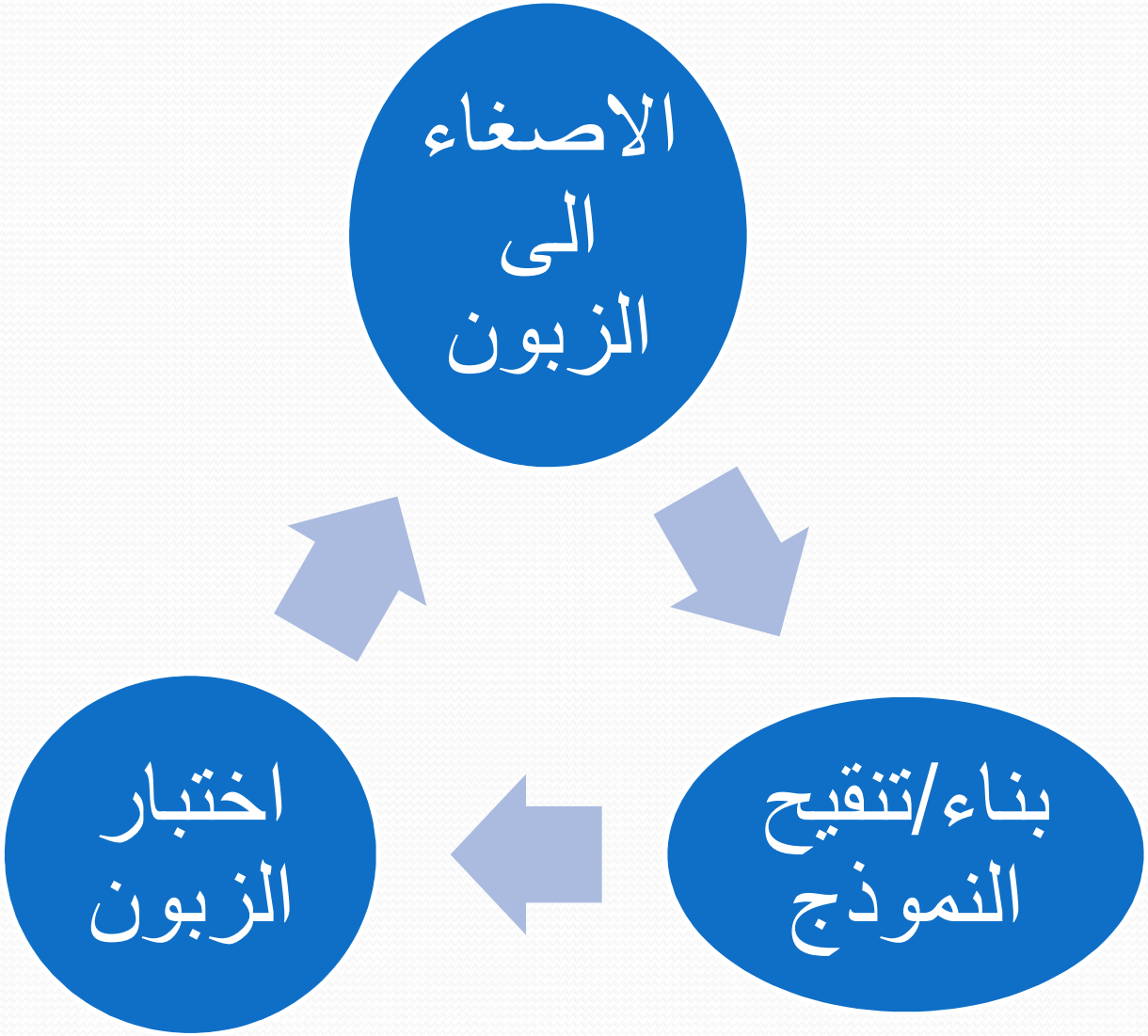
# نموذج "النمذجة الأولية" Prototype Model

- يبدا هذا النموذج من خلال تعريف الزبون مجموعة من الأهداف العامة للبرنامج، ولا يحدد بالتفصيل كل متطلبات الادخال أو المعالجة أو الاخراج،

- في بعض الحالات قد يكون مطور غير متأكد من فعالية الخوارزمية او النموذج، أو الشكل الذي يجب أن يأخذه تفاعل الإنسان مع الآلة، لذلك فان نموذج النمذجة الأولية (Prototyping Paradigm) جاء ليقدم الطريقة الفضلى في حل المشاكل في هذه الحالات وحالات كثيرة غيرها.

- الشكل التالي يبين نموذج النمذجة الاولية:

# نموذج "النمذجة الأولية" Prototype Model



# نموذج "النمذجة الأولية" Prototype Model

- خطوات هذه النموذج تكون كالاتي:

١. الاصغاء الى الزبون Listen to customer

- يبدا هذا النموذج بجمع المتطلبات النظام من خلال اجتماع يحصل بين مطور النظام والزبون لتعريف وتحديد الأهداف الإجمالية للبرنامج.

٢. بناء النموذج build/revise mock-up

- تطوير نموذج أولي (مبدئي ، تجريبي): استنادا إلى حاجات الزبون او المستخدم، حيث يعمل مصممو النظم على تأمين نموذج سريع بواسطة البرمجيات والأدوات المساعدة.

# نموذج "النمذجة الأولية" Prototype Model

٣. اختبار الزبون customer test drives mock-up

- تحديد إذا كان النموذج الأولي مقبولا أم لا (تجربة النموذج).

- في هذه المرحلة يعيد الزبون أو المستخدم تقييم النموذج الأولي، لضمان ان هذا النموذج قد لبى متطلبات تصميم النظام، وكذلك يساع المطور من ايجاد فهم افضل للمتطلبات الواجب تلبيتها للزبون.

- يحدث التكرار في هذه المرحلة لضبط النموذج الاولي ولضمان تحقيق متطلبات الزبون

# نموذج "النمذجة الأولية" Prototype Model

**- مميزات النمذجة الأولية:**

١. يمكن اعتبارها اقتصادية حيث تقل الكلفة بسبب تلافي الأخطاء والالتباس وسوء فهم الاحتياجات في مرحلة مبكرة.

٢. سهلة الفهم حيث تتجه الى بناء النظام خطوة بخطوة، ولا يتم الانتقال الى خطوة تالية إلا بعد التأكد من الخطوة الأولى.

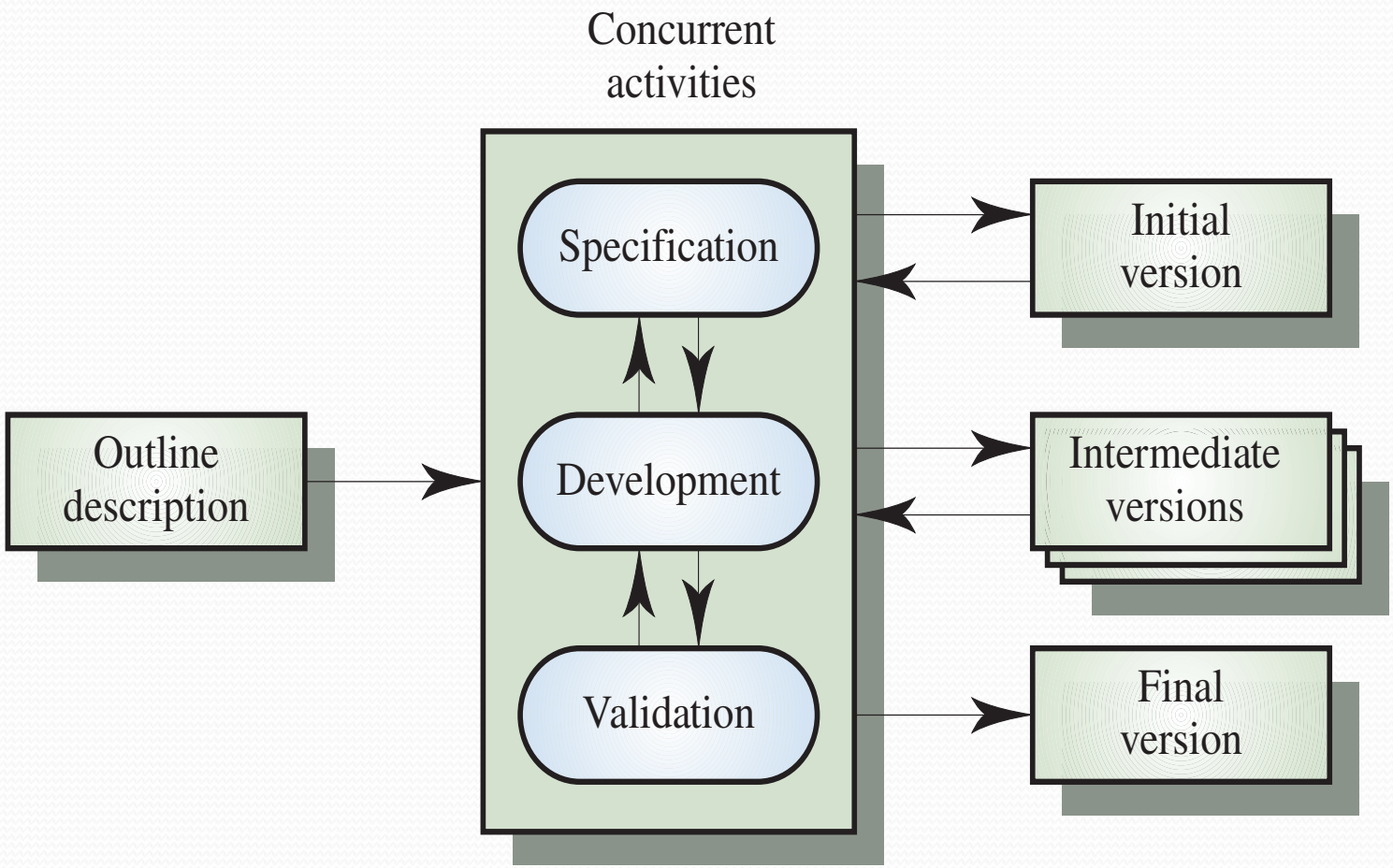
٣. يستوعب المستخدم النظام جيداً قبل تركيبه وتنفيذه بشكل نهائي.

**- عيوب النمذجة الأولية:**

١. معظم المشاريع بالكاد يكون اول نظام يبني قابلا للاستخدام فقد يكون بطئيا جدا او كبيرا جدا او صعب الاستخدام او الثلاثة معا.

٢. اذا لم يتم التحكم بالمراحل يمكن أن تطول عملية النمذجة.

# Prototype Model نموذج "النمذجة الأولية"



# النماذج التطورية Evolutionary Models

- هناك اعترافٌ متنامٍ بأن البرمجيات تتطور على مر الزمن. إذ تتغير متطلبات الزبون والمنتَج مع تَقدم عملية تطوير هذا المنتَج، وهذا ما يجعل المسار المستقيم ( النموذج الانحداري Waterfall Model) باتجاه إنتاجه غير واقعي. لقد صُمِّم النموذج التتابعي الخطي لحالات التطوير المباشر (خط مستقيم). وبمعنى آخر، تفترض هذه الطريقة التتابعية أن النظام كله سيسلَّم بعد اكتمال هذا التتابع الخطي.

- ومن جهة أخرى، فقد صُمِّم نموذج النمذجة الأولية (Prototype Model) لمساعدة الزبون (أو المطوِّر) على فهم المتطلبات، ولم يصمم عموماً لتسليم نظام نهائي. فلَم تُلاحَظ الطبيعة التطورية للبرمجيات في كلا هذين النموذجين التقليديين لهندسة البرمجيات.

# النماذج التطورية Evolutionary Models

- النماذج التطورية evolutionary models هي نماذج تكرارية، وتوصَّف بطريقة تمكِّن مهندس البرمجيات من تطوير نسخ أكثر تعقيداً من البرمجيات، وسنذكر فيما يلي اثنين من هذه النماذج:

١. **النموذج التزايدي (Incremental Model)**

- النموذج المُتزايد: الذي يجمع مكونات من النموذج المتتالي الخطي مطبقة بشكل متكرر، ويقوم النموذج المتزايد بعمل تتابع مخطوط بصيغة متتالية في الترتيب لكن مع تقدم زمن الإنتاج، فينتج كل تتابع خطي برمجيّات متزايدة جاهزة للتسليم.

- هذا النموذج كما يقترحه اسمه يعمل بالتزايد، أي أن التطبيق الناتج لا يُصدر مرة واحدة و إنما على دفعات تسمى بالمتزايدة.

# النماذج التطورية Evolutionary Models

- كمثال، لنأخذ تطبيق على أجهزة المكتب، هذا التطبيق يسمح لك بإضافة عملاء و بياناتهم و بيانات شرائهم كما يسجل كل البضائع في المخزن و يتابع عددها بإنشاء تقارير اسبوعية.

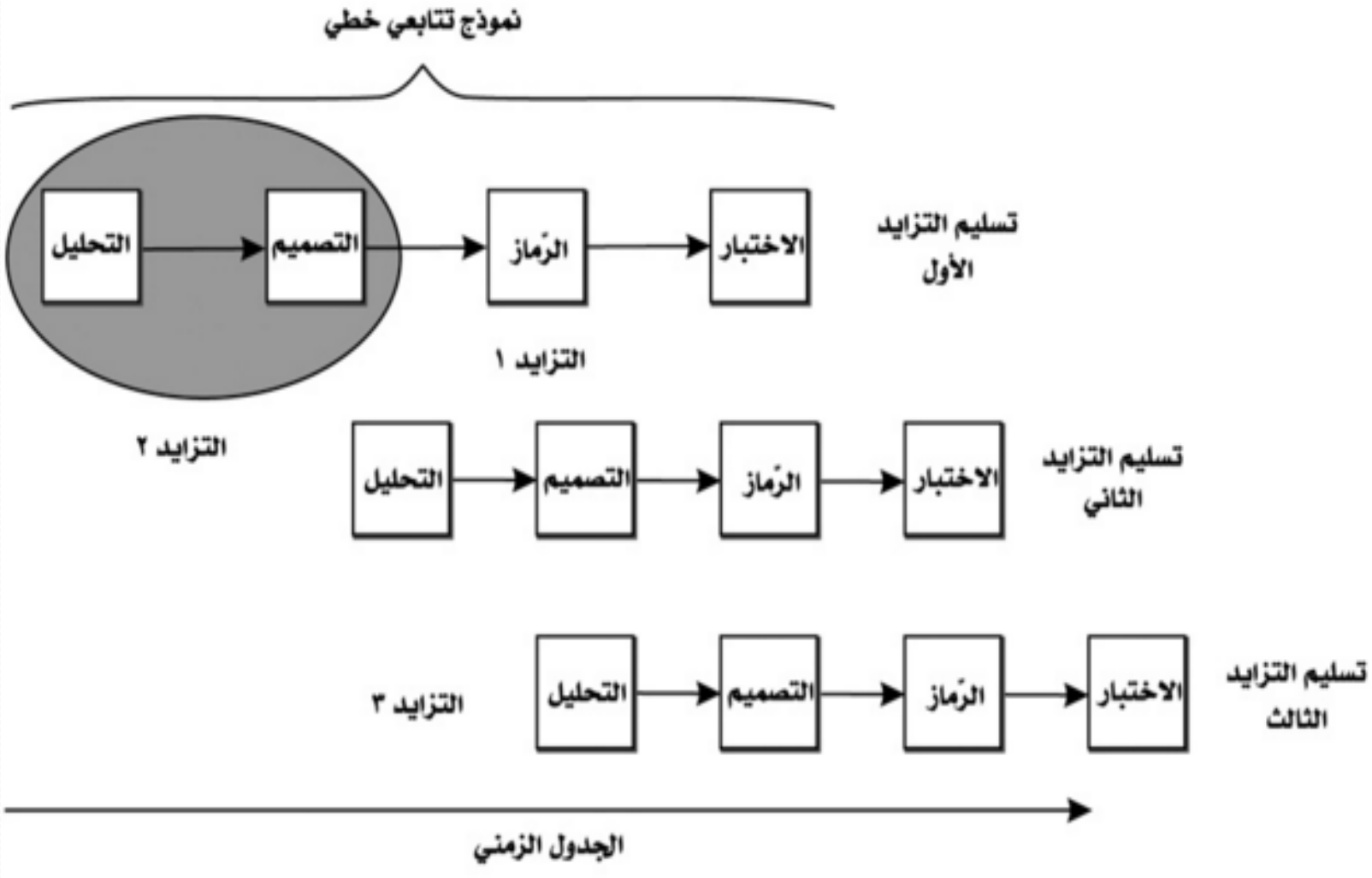
- إذا كان التطبيق سيتم تطويره باستخدام هذا النموذج، فإن أول عملية يقوم بها المطورون هو تقسيم الوظائف إلى مجموعات غير مترابطة، بحيث يتم تطوير كل مجموعة على حدة بشكل متزايدة، يتم البدء بأهم و أعقد الوظائف و كون تحديد وظيفة ما بأنها مهمة راجع لأسباب منها رغبة العميل نفسه ،

# النماذج التطورية Evolutionary Models

- ولنفترض أن إضافة العملاء هي أهم خاصية برأي العميل، ذلك يتم تطبيقها و اختبارها و اطلاقها في النظام و السماح للعميل لا لتجربتها فقط بل و البدء بالاستخدام الفعلي ، حالما تطلق الوظيفة يتم البدء حالاً بتطوير المتزايدة اللاحقة و بنفس المنظومة يتم اختبارها و من ثم دمجها مع الوظيفة السابقة واطلاقها للعميل و هكذا، اذاً نرى أن العميل له دور تفاعلي اساسي في هذا النموذج ، كما أن المطورون يحصلون على رأيه بشكل متتالي و سريع مما يسمح لهم بالتعديل بفاعلية اكبر.

- الشكل يبين النموذج التزايدي.

# النماذج التطورية Evolutionary Models



# النماذج التطورية Evolutionary Models

- **من فوائد هذه النموذج** انه

١. حقق تطورا فظيعا في تطوير البرمجيات
٢. انشئ مفاهيم جديدة، كما انه يتميز بملائمته للعصر الحالي
٣. الحصول على التطبيقات في وقت سريع حيث ان مدة المشاريع تصل الى شهور حينما تمتد مشاريع نموذج الشلال الى سنين .

- ولكن ان من عيوب هذا النموذج انه يطلق نسخ كثيرة من نفس التطبيق، هذا ربما يسبب الانزعاج لدى المستخدم و الذي لا يكون بالضرورة نفس شخص العميل ،

# النماذج التطورية Evolutionary Models

- في مثالنا السابق مثلا يكون العميل هو صاحب الشركة لكنه لا يستخدم النظام فعليا و انما موظفونه، ان الموظفون هم المستخدمون المقيقون لهذا التطبيق و كونهم لا يشترونه منك لا عني ان لا تضعهم في الحسبان ، لأن في النهاية الغرض من التطبيق دائما هو لتحسين و تطوير فاعلية العمل و حين يكون الموظفون منزعجون فإن فاعليتهم ستقل و سينزعج منك العميل في النهاية.

- كن هذا العيب يمكن اصلاحه بسهولة بتقسيم مناسب للوظائف بحيث في كل دفعة يتوجه الاهتمام لقسم معين ، او بالتدريب المناسب لموظفين حال اطلاق نسخة حديدة بحيث لا يشعرون بالضياع ، أو أسوأ: أن يشعروا بالغباء و هو السبب الحقيقي غالباً وراء كل انزعاج من المستخدم .

# النماذج التطورية Evolutionary Models

٢. النموذج اللولبي او الحلزوني Spiral Model

- هو نموذج تطوري لعملية البرمجة يقرن الطبيعة التكرارية للنمذجة الأولية بالنواحي النظامية والمحكومة للنموذج التتابعي الخطي
- يقدم إمكانية تطوير سريع لنسخ تزايدية من البرنامج
- يكون الإصدار التزايدي خلال التكرار الأولي نموذجا ورقيا أو نموذجا أوليا
- ينقسم النموذج الحلزوني إلى عدد من نشاطات الهيكل تسمى منطقة المهمة task region هناك عادة ما بين ٣ إلى ٦ مناطق وهي:

# النماذج التطورية Evolutionary Models

١. الاتصال بالزبون Contact the Customer (التواصل الفعال بين الزبون والمطور)،
٢. التخطيط Planning (تعريف الموارد والمسارات الزمنية للمشروع)،
٣. تحليل المخاطرة risk analysis ( قييم المخاطرة التقنية والإدارية)،
٤. الهندسة Engineering (بناء تمثيل أو أكثر للتطبيق)،
٥. البناء والإصدار construction & release (مهمات البناء والاختبار)،
٦. تقويم الزبون customer evaluation (الحصول على تقييم الزبون للعمل).

# النماذج التطورية Evolutionary Models

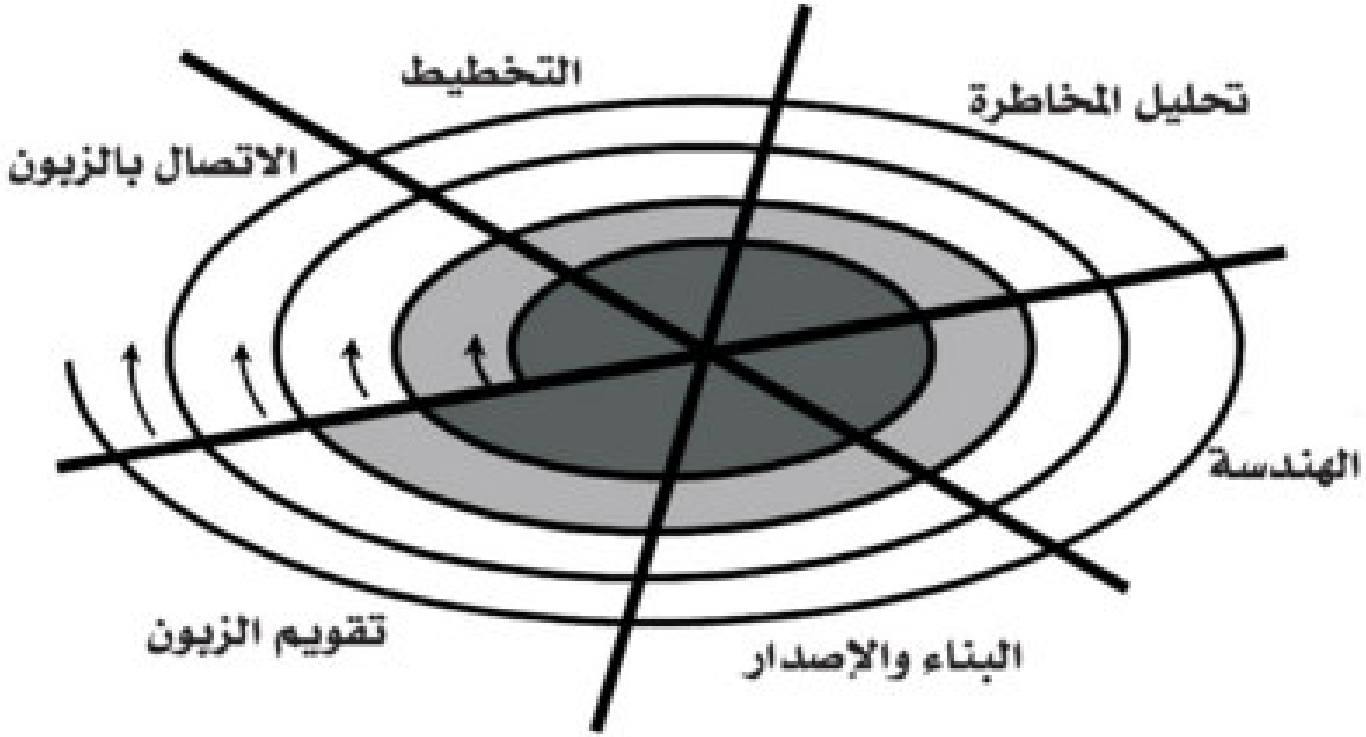
- حالما تبدأ عملية البرمجة التطورية يتحرك فريق هندسة البرمجيات حول الحلزون باتجاه عقارب الساعة بدءا من النواة
- خلافا للنماذج التقليدية التي تنتهي بتسليم البرنامج يمكن تكييف النموذج الحلزوني لتطبيقه على امتداد كامل حياة البرنامج
- يستخدم النموذج الحلزوني النمذجة الأولية كآلية لتقليل المخاطر التقنية ولكنها تمكن المطور من تطبيق النمذجة الأولية في أي مرحلة من مراحل تطور المنتج

**العيوب:**

النموذج الحلزوني "**قلل من المشاكل بشكل كبير**" **ولكن** يمكن أن تظهر نفس المشكله في التي كانت تظهر في الشلال حيث في حال أكتشف المبرمج أن هناك خطأ أو نقص في التصميم لا يمكن العودة للخلف ويجب الانتظار حتى البدء في الدورة الثانية

# النماذج التطورية Evolutionary Models

- الشكل الاتي يبين النموذج اللولبي او الحلزوني



# تصميم وبرمجة واجهة المستخدم

# DESIGN AND PROGRAMMING USER INTERFACE

د.أيمن حمارشه

# واجهة المستخدم
# User Interface

في أيامنا هذه دخلت الحوسبة في حياتنا بوتيرة متسارعة بحيث أصبح استخدام الأجهزة الالكترونية المختلفة جزءاً لا يتجزأ من حياة كل واحد منا. من مظاهر هذه الحوسبة وجود الحواسيب الشخصية (Personal Computers) في البيوت, في الشركات, في المؤسسات التجارية والمالية وغيرها. وكذلك اقتناء الهواتف النقالة التي أصبحت عبارة عن حواسيب متنقلة يمكنها إنجاز الكثير من الوظائف والمهمات وخاصة تخزين البيانات ومعالجتها بالإضافة للوظائف التقليدية للهاتف العادي.

هناك شكل آخر للحوسبة هو الأجهزة المنزلية الحديثة المزودة بنظم رقمية (Digital Systems) للتحكم في عمل هذه الأجهزة كالثلاجات والغسالات وأفران المايكروويف وأجهزة التكييف وغيرها.

هذا الكم الهائل من الأجهزة يجعل من الصعب على المستخدم العادي أن يكون ملما بكافة التفاصيل اللازمة بكيفية استخدام هذا الجهاز أو ذاك. ويلجأ المستخدم عادةً لقراءة كتيِّبات الاستخدام (Manuals) المصاحبة لهذه الأجهزة -والتي يمكن أن تتكون من عدد ليس قليل من الصفحات- لذلك ليس مستغربا أن نجد أن أغلب المستخدمين يجهلون كيفية استخدام هذه الأجهزة بالشكل الأمثل وفي أحسن الحالات يستخدمون الجزء القليل من إمكانيات هذه الأجهزة. لذلك فإن الشركات المنتجة لهذه الأجهزة تسعى لإقناع الزبون المحتمل بالفوائد التي سوف يجنيها عند اقتنائه هذا المنتج من خلال ما يسمى واجهة المستخدم (User Interface) التي تلعب دورا كبيرا في مساعدة المستخدم على فهم الكثير من وظائف الجهاز بل وكيفية الاستخدام أيضا.

لذلك تسعى هذه الشركات لتصميم واجهات تساعد المستخدم على فهم وظائف هذه الأجهزة وعلى إدراك كيفية التعامل معها وفي وقت قصير. فالمستخدم مثلا لن يعجبه فرن المايكروويف الذي يحتوي على لوحة رقمية (Digital Panel) مملوءة بالمفاتيح والأزرار المختلفة التي يجب عليه معرفة وظيفة كل منها ليتمكن من استخدام هذا الفرن, ولكنه سوف يكون سعيدا إذا كان هذا الفرن يحتوي فقط على شاشة صغيرة (Display) ومؤقت (Timer) وزر تشغيل وما عليه سوى استخدام المؤقت لتحديد فترة التشغيل ثم

الضغط على زر التشغيل ومراقبة العملية من خلال الشاشة لا أكثر ولا أقل. بمعنى أن المستخدم يفضل دائماً اقتناء الأجهزة التي تتميز بالبساطة وبسهولة التعامل معها. وقياسا على ذلك نجد أن البرامج التي تدير هذه الأجهزة وتتحكم في عملها يجب أن تكون أيضا سهلة, بسيطة ومفهومة للمستخدم, وما ينطبق على هذه الأجهزة ينطبق على الحواسيب الشخصية أيضا. فاستخدام الحواسيب كان صعبا نسبيا عندما كانت تُستخدم واجهات النمط النصي (Text Mode) والتي كانت تفرض على المستخدم أن يكتب الأوامر مستخدما لوحة المفاتيح وهذا يسمى المواجهة الخطية (Command Line) . هذا النمط يتميز بأن على المستخدم حفظ كميات كبيرة من الأوامر والحرص دائما على كتابة هذه الأوامر بدون أي أخطاء إملائية أو قواعدية, ثم تحسن الأمر بعض الشيء مع ظهور الواجهات التي تسمح للمستخدم اختيار الأوامر من خلال قوائم (Menu) تظهر أمامه على الشاشة.

تطورت الأمور كثيرا في مرحلة لاحقة مع ظهور نوع جديد من واجهات المستخدم هي واجهة المستخدم الرسومية GUI (Graphical User Interface) التي يتعامل فيها المستخدم مع رسومات صغيرة تسمى أيقونات (Icons) يقوم المستخدم من خلالها بتوجيه الأوامر للحاسوب وذلك بالنقر بواسطة الفأرة (Mouse) على أي من هذه الأيقونات لتنفيذ المهمة التي يريدها. وقد عملت هذه الواجهات على جعل عملية تفاعل المستخدم مع الحاسوب سهلة ومريحة. فمثلاً عندما تظهر أمام المستخدم أيقونة على شكل زر وقد كتب على هذا الزر أمر مثل نعم أو موافق أو خروج فإنه لن يتردد في النقر فورا على أحد هذه الأزرار لإحداث تأثير ما أو تنفيذ أمر معين. وعند حدوث خطأ ما فسوف تظهر على الشاشة رسالة قصيرة توضح المشكلة وأحيانا تحتوي الرسالة على معلومات تخبره ماذا عليه أن يفعل لحل المشكلة.

في البرامج الكبيرة والمعقدة سيكون من الصعب على المستخدم تصور كافة الإمكانيات التي يملكها هذا البرنامج وذلك لاستحالة إظهار جميع هذه الإمكانيات من أزرار ومربعات اختيار ورسومات مختلفة على الشاشة في نفس الوقت. في هذه الحالات يتم اللجوء إلى هيكلية القوائم (Menu Structure) التي يتم من خلالها استخدام مساحة الشاشة بشكل جيد وحيوي.

تقوم الفكرة الأساسية في استخدام القوائم على إظهار الكثير من الوظائف التي يمكن للنظام أو البرنامج القيام بها وبشكل مختصر. على سبيل المثال إذا أراد المستخدم إدخال صورة في ملف نصي ولا يعرف كيف يفعل ذلك ولكنه يعرف أن البرنامج يتيح له هذه الإمكانية فإنه سوف يبدأ في البحث عن الخيار الذي يتيح له ذلك ضمن مجموعة كبيرة من الخيارات (Options) التي يمكن للبرنامج القيام بها والتي قد يصل عددها إلى المئات, وحتما سوف يجد ذلك الخيار تحت اسم "إدخال" (Insert) . ولو تخيلنا للحظة أن هذه الخيارات كلها سوف تظهر على الشاشة بالتتابع على شكل قائمة واحدة عندها سيكون على المستخدم أن يمر على جميع هذه الخيارات حتى يصل إلى الخيار المنشود. ولكن لحسن الحظ لا يكون البحث بهذا الشكل ولا يجب على المستخدم فعل ذلك لأن طريقة البحث تكون بشكل مختلف ولها مسار مختلف, والسبب أن الشكل الذي تظهر فيه الخيارات قد صمم بشكل مختلف.

في أغلب الواجهات الحديثة يتم تصميم شريط يسمى شريط القوائم (Menu Bar) تظهر عليه مجموعة من القوائم الرئيسية يتراوح عددها عادةً بين 6 و10 فقط, وعند فتح أي قائمة من هذه القوائم الرئيسية سوف نجد أنها تحتوي على مجموعة من الوظائف أو المهام التي ترتبط مع بعضها البعض بشكل أو بآخر وتظهر على التوالي بشكل عمودي. هذه القائمة الرئيسية تتفرع بدورها إلى قوائم فرعية (Sub Menu) لذلك فإن المستخدم الذي يريد إدخال صورة لن يتوقف عند خيار ملف (File) أو خيار تحرير (Edit) مثلا, بل بديهيا سوف يتوقف عند خيار إدخال (Insert) وسوف ينقر هذا الخيار وعندها سوف تظهر أمامه قائمة بالأشياء التي يمكن إدخالها, وبشكل تلقائي سوف يستعرض المستخدم محتويات هذه القائمة ولن يختار رقم الصفحة (Page Number) أو رمز (Symbol) وإنما سيختار صورة (Picture). عند النقر على خيار صورة سوف يرى قائمة فرعية أخرى توضح له كيفية إدخال صورة من أكثر من مصدر, وفي النهاية سوف يصل المستخدم إلى هدفه حتى لو لم يكن على معرفة مسبقة بكيفية القيام بهذه المهمة ولن يأخذ منه هذا جهدا كبيرا أو وقتا طويلا.

إن مهمة مصممي واجهة المستخدم الأساسية هي إنشاء قوائم تتسم بالوضوح التام بحيث تكون المهمات والوظائف المطلوبة ظاهرة أمام المستخدم بشكل لا يجعله يهدر الكثير

من الوقت في البحث عنها وأن يتم بناء هذه القوائم على نحو يحقق الوصول إلى المهمات بخطوات متتالية. أما مهمة المبرمجين فهي تطوير البرنامج بشكل يسمح بإنشاء هذه القوائم مهما كانت درجة الصعوبة بحيث تعمل هذه القوائم بالشكل المطلوب والمبرمج الخبير البعيد النظر يعلم بأنه من الصعب كتابة البرنامج بشكل نهائي من أول مرة, هذا من ناحية, من ناحية أخرى تطوير البرنامج بحيث لا تكون عملية الوصول مقتصرة على طريق واحد فقط (من خلال الفأرة مثلا) بل أن يكون ذلك ممكنا أيضا بطرق أخرى (من خلال مفتاح Enter وأسهم الانتقال في لوحة المفاتيح).

# تصميم واجهة المستخدم

# User Interface Design

إن الحقيقة الأساسية التي يجب أخذها بعين الاعتبار عند تطوير التطبيقات (Applications) المختلفة هي أن واجهة التطبيق أهم شيء بالنسبة للمستخدم, وذلك لأن المستخدم لا يرى من هذا التطبيق سوى واجهته. ومن الطبيعي أن المستخدم يريد واجهات تلبي احتياجاته المختلفة وعلى رأس هذه الاحتياجات سهولة الاستخدام إلا أنه من الملاحظ أن الكثيرين من مطوري التطبيقات لا يعيرون اهتماما كافيا لهذا الجانب ويكون جل اهتمامهم منصبا على وضع شفرات وأكواد "ذكية" أو تصميم نظام ألوان جذابة بدلا من التفكير في كيفية جعل التطبيق أكثر سهولة في الاستخدام. لذلك تعتبر واجهة المستخدم الجيدة هي تلك التي تسمح للأشخاص الذين يستخدمون التطبيق باستخدام ميزات وخصائص التطبيق دون الحاجة إلى قراءة كتيبات استخدام (Manuals) تتكون من عشرات الصفحات أو الحصول على دورات تدريب خاصة بكيفية الاستخدام.

مما سبق يمكن القول أنه لكي تلقى الواجهة نجاحا وتصبح مرغوبة لدي المستخدم فإنه يجب أن تتوفر فيها ميزات وخصائص منها على سبيل المثال الوضوح بحيث يفهم المستخدم بديهيا ما عليه القيام به للوصول إلى ما يريد وهذا يحصل عندما تكون الواجهة مزودة برموز ونصوص مفهومة تقود المستخدم بخطوات متتالية إلى هدفه. ومن هذه الخصائص أيضا سهولة التعلم والتدريب على استخدام الواجهة. ولغرض الوصول إلى واجهات تتوفر فيها هذه الميزات وغيرها فإنه يجب الالتزام ببعض القواعد وكذلك استخدام بعض التقنيات ومنها:

الاتساق أو الانسجام (Consistency): وهذا يعني أن تعمل الواجهة على نفس النسق بمعنى أن أي حدث معين يجب أن تكون له نفس النتيجة وبحيث يفهم المستخدم أن تكرار هذا الحدث ولكن مع عنصر آخر في الواجهة سيكون له نفس الأثر. فمثلا النقر المزدوج على أيقونة معينة تمثل مجلدا أو ملفا سيؤدي إلى فتح هذه الأيقونة وعرض محتوياتها وهذا ما يجب أن يحدث في كل مرة يتم فيها النقر المزدوج على أية أيقونة مهما كان التطبيق أو البرنامج الذي تمثله الأيقونة. وبنفس الطريقة يجب أن تكون وظائف العناصر المتشابهة التي تظهر على الواجهة هي

نفسها, فمثلا النقر على الزر X الموجود على شريط العنوان في أية نافذة في نظام Windows يؤدي إلى إغلاق التطبيق أو البرنامج وهذا ما يجب أن يحدث عند النقر على نفس الزر في نافذة أخرى لتطبيق آخر. هذا –طبعا- يتطلب وضع الأزرار في جميع النوافذ في نفس المكان وكذلك استخدام نفس الصيغة في التسميات (Labels) والرسائل (Messages) بالإضافة إلى استخدام نفس الألوان (Color Schemes) في الأماكن المختلفة. إتباع هذه الخاصية عند التصميم يُمَكِّن المستخدم من تكوين نموذج ذهني دقيق لطريقة عمل عناصر الواجهة مما يساعد على سرعة الفهم و التعلم.

وضع معايير تصميم ثابتة (Set Modeling Standards): إن الطريقة الوحيدة التي يمكن من خلالها تحقيق خاصية الاتساق في واجهة المستخدم هي وضع معايير ثابتة للتصميم ومن ثم إتباع هذه المعايير بدقة وخاصة تلك المعايير التي تم استخدامها سابقا في تطوير البرمجيات بشكل عام وواجهات المستخدم بشكل خاص وهو ما يسمى نمذجة معايير التطبيق ( Modeling Standards).

في بعض الأحيان عند تطوير واجهات لبعض الأنظمة والتطبيقات يقوم أصحاب هذه الأنظمة بتقديم بعض الأفكار والمقترحات التي قد تكون غير عادية أو ربما غير مناسبة فيما يتعلق بالكيفية التي يجب أن تكون عليها هذه الواجهة أو طريقة عملها. في مثل هذه الحالة يجب الاستماع لهذه الأفكار ولكن في الوقت نفسه يجب تقديم التوضيحات والبراهين على صواب المعايير والطرق التي يستخدمها المطورون وأنها في نهاية المطاف تصب في مصلحة النظام أو التطبيق.

شرح قواعد الاستخدام (Explain the rules): يعتبر شرح كيفية استخدام الواجهة للأشخاص الذين سوف يقومون بالتعامل مع التطبيق أمرا ضروريا. وهنا تبرز أهمية امتلاك الواجهة لخاصية الاتساق حيث أنه يمكن شرح قواعد الاستخدام مرة واحدة فقط كما أنه لا داعي لشرح التفاصيل كلها لكونها تتكرر في أماكن عدة مما يجعل من السهل على المستخدم تعلم كيفية التعامل مع الواجهة في وقت قصير وجهد قليل.

**التنقل بين عناصر الواجهة ( Navigation between user interface items): يجب أن يكون التنقل بين العناصر الرئيسية المكونة للواجهة سهلا وواضحا لأن المستخدم سوف يصاب بالإحباط وربما لن يعود لاستخدام الواجهة مرة أخرى إذا كان الانتقال من شاشة إلى أخرى صعبا مثلا. من ناحية أخرى إذا كان التنقل بين عناصر الواجهة المختلفة منسجما مع المهمات والوظائف التي يقوم المستخدم بإنجازها فإن هذا سوف يساعد المستخدم على فهم وإدراك خصائص التطبيق بشكل أفضل. وبما أن المستخدمين مختلفون في طريقة عملهم فإن النظام يجب أن يكون مرنا بما فيه الكفاية لكي يكون قادرا على دعم هذه الطرق المختلفة وذلك من خلال تطوير ما يعرف بمخطط تدفق الواجهة (User Interface Flow Diagram).**

**التنقل داخل الشاشة (Navigation within a screen): تتميز المجتمعات المختلفة باختلاف ثقافتها وطريقتها في التعامل مع الأشياء. فالمجتمعات الغربية تختلف عن بعض المجتمعات الشرقية ومنها العربية في طريقة القراءة والكتابة مثلا, حيث نجد أن الإنسان الأوروبي متعود على القراءة والكتابة من اليسار إلى اليمين ومن الأعلى إلى الأسفل ونجد الإنسان العربي متعود على القراءة والكتابة من اليمين إلى اليسار ومن الأعلى إلى الأسفل, أما في الصين مثلا فهم يكتبون ويقرؤون من الأعلى إلى الأسفل. هذا التنوع يجب أن ينعكس على الطريقة التي يتم بها تصميم الواجهة. فالواجهة الموجهة للاستخدام من قبل شخص أوروبي يجب أن يكون التعامل فيها مع الاتجاهات والتنقل وكتابة النصوص منسجما مع ما تعود عليه هذا المستخدم حيث سيكون صعبا عليه التعامل مع الاتجاه إذا كان من اليمين إلى اليسار وكذلك الأمر مع المستخدم العربي الذي تعود على أن يكون الاتجاه من اليمين إلى اليسار أي أن التنقل داخل الشاشة يجب أن يكون بشكل متوافق مع ثقافة المستخدم وطريقته.**

**كتابة الرسائل والتسميات بشكل فعال ( Word messages and labels effectively): إن الكتابة التي تظهر على الشاشة تعتبر المصدر الرئيسي للمعلومات بالنسبة للمستخدم, لذلك يجب أن تكون طريقة كتابة التسميات والرسائل التي تُوَجَّه للمستخدم واضحة ومفهومة وأن يتم صياغة التعبير بشكل يجعله سهل الفهم من قبل المستخدم كاستعمال الجمل الواضحة والكلمات الكاملة بدلا من استعمال الاختصارات والرموز والجمل المبهمة. لذلك إذا كان التعبير ضعيفا فلن يتم فهمه جيدا من قبل المستخدم, أما الرسائل التي يوجهها النظام للمستخدم**

فيجب صياغتها بشكل واقعي وعلى نحو يضمن للمستخدم التحكم بشكل فعال وصحيح في العمليات التي يريد من النظام أو التطبيق أن يقوم بها. فمثلا الرسالة التي نصها "إدخال معلومات شخصية" لن يكون لها نفس وضوح الرسالة التي نصها "إدخال الاسم الثلاثي" حيث ستكون الرسالة الأولى

مبهمة بعض الشيء بالنسبة للبيانات التي يجب إدخالها, أما الرسالة الثانية فهي واضحة جدا وسيقوم المستخدم بإدخال البيانات المطلوبة بشكل صحيح. بالإضافة إلى ذلك يجب أن تعرض الرسائل باستمرار وفي مكان مناسب على الشاشة.

الفهم الصحيح لدور مكونات الواجهة (Understand the UI widgets): يقصد بهذا أن يتم استخدام كل مكون من مكونات الواجهة على الوجه الصحيح وعلى النحو الذي يحقق الغرض من وجود هذا المكون, لذلك يجب تعلم كيفية استخدام كل مكون وكل عنصر من خلال معرفة الوظيفة التي يقوم بها.

دراسة واجهات أنظمة وتطبيقات أخرى (Look at other UI applications): من المفيد أحيانا النظر بعمق إلى واجهات أنظمة وتطبيقات أخرى والاطلاع على الأفكار المستخدمة في تصميمها ومحاولة الوصول إلى كل ما هو جديد ومبتكر ومحاولة الاستفادة من ذلك وفي الوقت نفسه محاولة معرفة الجوانب السلبية في هذه الواجهات حتى لا يقع المصمم في نفس الخطأ مرة أخرى عند تصميم الواجهات الخاصة به وأن لا يقوم بتقليد التصاميم الغير جيدة والغير ناجحة.

استخدام الألوان (Use color appropriately): تلعب الألوان دورا مهما في تصميم الواجهات سواءً من ناحية إضفاء مسحة جمالية على الواجهة أو من خلال توظيفها في إبراز بعض العناصر في الواجهة. فمثلا يستخدم اللون الأحمر في تحذير المستخدم أو لفت انتباهه, ويتم اختيار ألوان أخرى للقيام بأدوار معينة كإبراز بعض عناصر الواجهة وكذلك على تحديد وفهم الوظائف المختلفة لعناصر الواجهة الأخرى. ومع ذلك يُنصَح بعدم الإفراط في استخدام الألوان بحيث يكون عدد الألوان المستخدمة مناسباً وكذلك عدم استخدام الألوان المرهقة للنظر بكثرة والشيء المهم الآخر هو أن تكون الألوان منسجمة وفي تناغم مع بعضها البعض وأن لا تُشعِر المستخدم بالنفور بل بالراحة وأخيرا يجب أن تضفي الألوان مسحة جمالية تعطي الواجهة شكلا جميلا وجذابا.

**إتباع قاعدة التباين (Follow the contrast rule): عند استخدام الألوان في واجهة التطبيق يجب التأكد أن الألوان لن تطغى على النص بحيث تجعله غير واضح أو غير مقروء. أفضل طريقة لفعل ذلك هي إتباع قاعدة التباين بحيث يتم اختيار لون خط غامق في كتابة النص واختيار**

**خلفية فاتحة ليكتب عليها أو العكس, فالنص المكتوب بلون أزرق على خلفية بيضاء سيكون واضحا ومن السهل قراءته في حين ستكون قراءة نفس النص لو كان على خلفية حمراء أمرا صعبا.**

**توقع أخطاء المستخدم (Except User's mistakes ): من المعروف أنه مهما كانت خبرة المستخدم كبيرة في التعامل مع التطبيقات فإن الخطأ البشري الغير مقصود وارد الحدوث. لذلك عند تصميم الواجهة يجب التفكير في استحداث الطرق التي تمنع أو تحد من وقوع هذه الأخطاء كما هو حاصل مثلا عند محاولة حذف ملف حيث يقوم النظام بسؤال المستخدم لتأكيد الأمر أو نفيه للتأكد من أن الأمر لم يصدر بطريق الخطأ.**

**قابلية التصميم للتخمين (Intuitable Design): بكلمات أخرى إذا كان المستخدم لا يعرف كيف يستخدم التطبيق فالتصميم الجيد للواجهة يساعد المستخدم على توقع أو تخمين ما يجب عليه فعله لتنفيذ شيء ما.**

**الكثافة الإجمالية للشاشة (Overall screen density): من الصعب على المستخدم فهم كيفية استخدام الواجهة إذا كانت الشاشة مزدحمة بالرموز والتسميات والصور المختلفة. ومن المتعارف عليه أن نسبة إشغال الشاشة بشكل عام يجب أن لا تتجاوز 40% من حجم الشاشة الكلي.**

**تجميع العناصر ( Grouping Items): من الأمور المهمة في تصميم الواجهات هو أن يتم تجميع العناصر التي ترتبط منطقيا مع بعضها البعض وذلك لتسهيل عملية وصول المستخدم إليها وذلك لأنه عادة يتم استخدام هذه العناصر مجتمعة عند تنفيذ مهمة معينة.**

**قابلية الواجهة للتطوير (UI Development): عند تصميم الواجهة يجب الأخذ بعين الاعتبار إمكانية تطوير هذه الواجهة مستقبلا وذلك لتلبية احتياجات المستخدم التي قد تنشأ لاحقا.**

## مبادئ تصميم واجهة المستخدم

## User Interface Design Principles

من أجل الوصول إلى تصميم جيد وناجح لواجهة المستخدم يراعي القواعد والتقنيات السابق ذكرها فإن على المصممين إتباع بعض المبادئ المهمة ومن أهمها:

1. الهيكلية (The structure principle): وهذا يعني تنظيم واجهة المستخدم بشكل هادف وبطرق مجدية ومفيدة مبنية على أساس نماذج واضحة ومتسقة بحيث تكون هذه النماذج مرئية يمكن للمستخدم تمييزها بسهولة, كما ينبغي وضع الأشياء التي ترتبط مع بعضها البعض في مجموعات وفصل الأشياء التي لا ترتبط مع بعضها. بشكل عام يمكن القول أن مبدأ الهيكلية يهتم بمعمارية واجهة المستخدم User Interface Architecture .
2. البساطة (The simplicity principle): حيث يجب أن يجعل التصميم المهمات سهلة في الفهم والتنفيذ وأن يسهل عملية التواصل مع المستخدم وذلك من خلال التعامل مع هذا المستخدم باللغة التي يفهمها وبالطريقة التي يفضلها. من الأمثلة على ذلك توفير طرق مختصرة Shortcuts تسهل عملية الوصول إلى تطبيقات Applications وإجراءات Procedures كبيرة.
3. الرؤية أو الشفافية (The visibility principle): ينبغي على التصميم الجيد إبقاء جميع الخيارات والموارد المطلوبة لتنفيذ مهمة معينة مرئية وواضحة أمام المستخدم وفي الوقت نفسه عدم تشتيت المستخدم بمعلومات غريبة وزائدة عن الحاجة. التصاميم الجيدة هي تلك التي لا تقدم للمستخدم كم هائل من المعلومات البديلة ولا تخلط المعلومات الضرورية بالمعلومات التي لا يحتاجها المستخدم في تنفيذ المهمة الآنية.
4. التغذية المرتدة (The feedback principle): حيث يجب على التصميم العمل على أن يبقى المستخدم على علم بجميع الإجراءات والتفسيرات المتعلقة بالمهمة المطلوب تنفيذها وذلك عن طريق تزويده وبشكل مستمر بكافة المعلومات المتعلقة بالتغيرات والشروط الجديدة التي قد تحدث أثناء التنفيذ وكذلك الأخطاء والاستثناءات ذات الصلة بالعملية والتي تهم المستخدم, وهذا يجب أن يكون بلغة واضحة لا لبس فيها, موجزة ومألوفة لدى المستخدم.

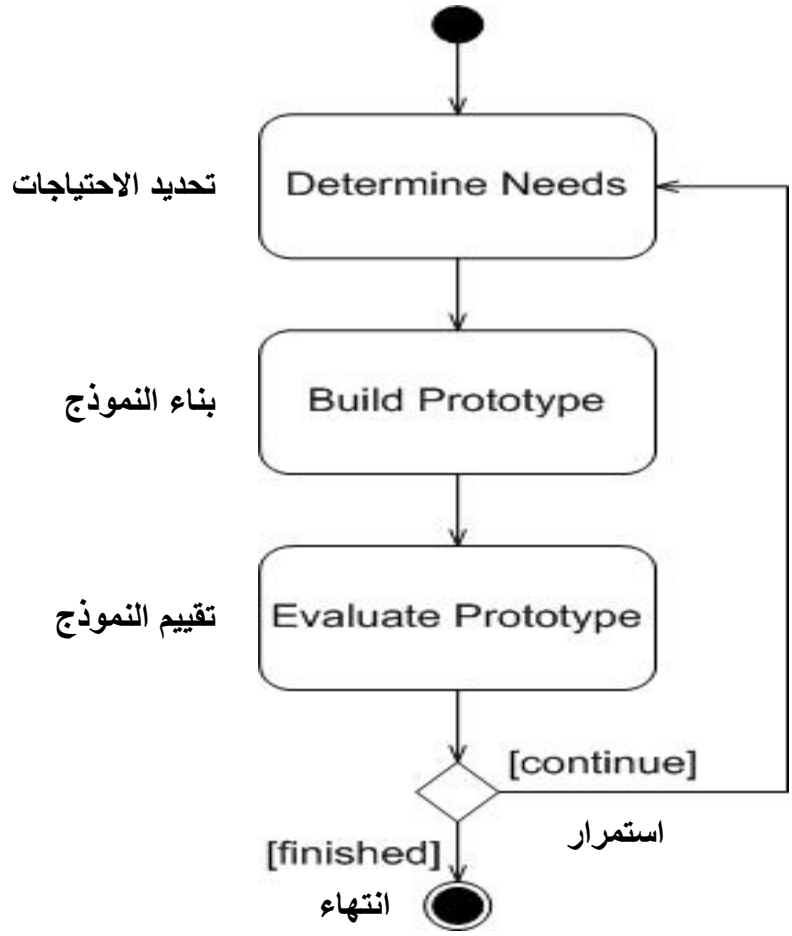
5. السماح (The tolerance principle): أي أن يكون التصميم مرنا بحيث يقلل من قيمة الأخطاء التي قد تحدث بسبب قلة خبرة المستخدم أو سوء استخدامه لموارد التطبيق وذلك من خلال السماح له بالتراجع وإعادة الأمر مرة أخرى ومنع حدوث الأخطاء إذا أمكن.

6. إعادة الاستخدام (The reuse principle): عندما يجعل التصميم المستخدم قادرا على إعادة استخدام مكونات الواجهة وعناصرها المختلفة فإن هذا يقلل من حاجة المستخدم للتذكر أو التفكير.

## نماذج واجهة الاستخدام

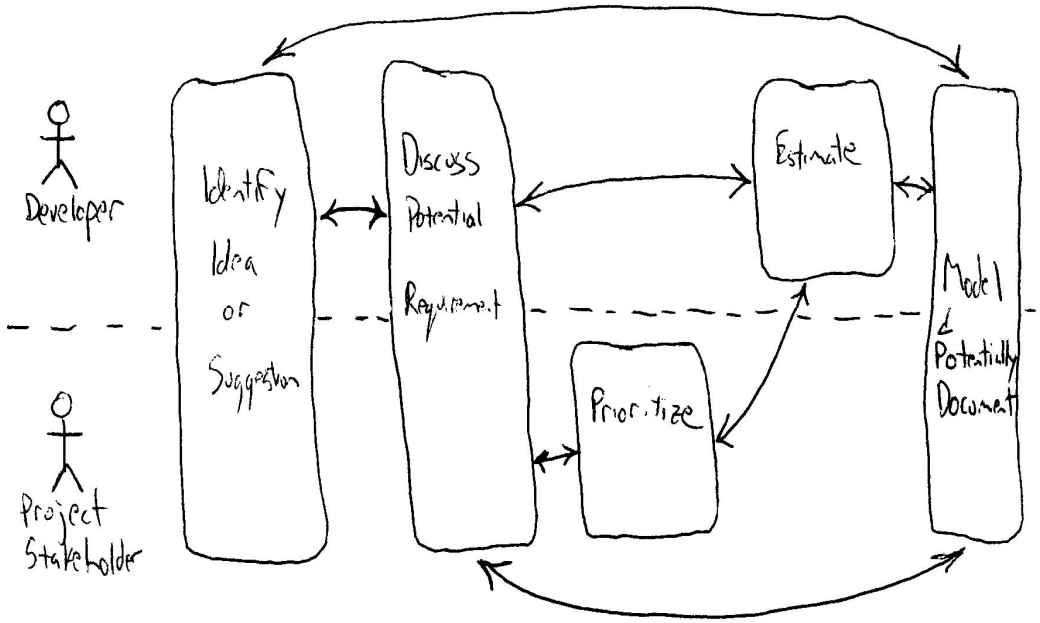
## User Interface Prototypes

يعتبر استخدام النماذج من أهم التقنيات المستخدمة في تصميم واجهات المستخدم وذلك لأنها تمنح المصمم إمكانية إنشاء أكثر من نموذج لنفس الواجهة ومن ثم دراسة وتحليل الجوانب المختلفة لكل نموذج من هذه النماذج من خلال إضافة أو حذف المكونات والعناصر المختلفة وصولا إلى النموذج الذي يمكن اعتباره الأفضل من بينها جميعا. بالإضافة لذلك فإن هذه النماذج تعطي المصمم تصورا أوليا عن واجهة المستخدم التي سوف يتم إنشاؤها, لذلك تعتبر هذه النماذج الأساس النظري أو يمكن القول أنها الخطوة الأولى والمهمة التي يبدأ منها تصميم وتطوير واجهة المستخدم الحقيقية.




الشكل (1) خطوات إنشاء نموذج أولي لواجهة المستخدم

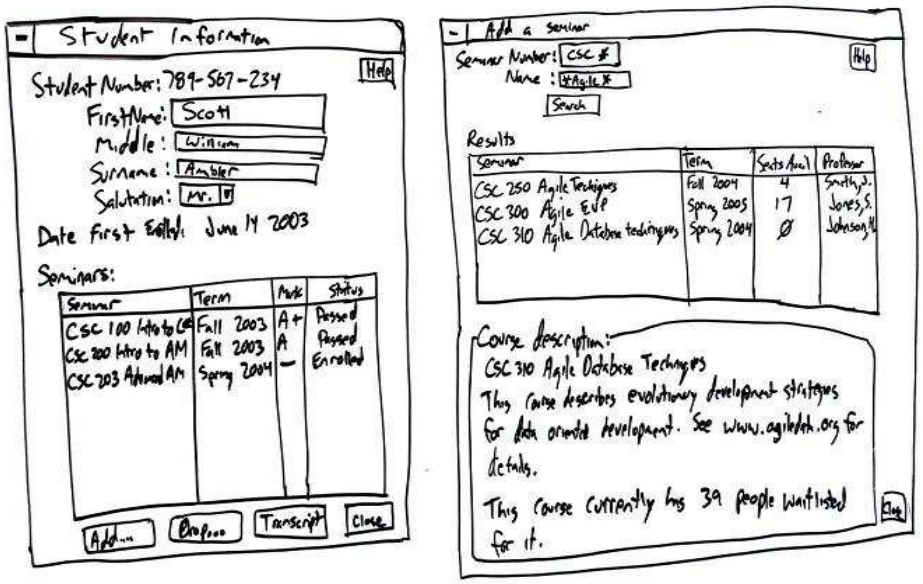
الشكل المبين أعلاه يُظهر خطوات عملية إنشاء نموذج واجهة المستخدم وكما هو واضح فإن أولى هذه الخطوات هي تحديد المتطلبات (Determine Needs) وهذا يعني تحليل الواجهة من حيث تحديد متطلبات واحتياجات المستخدم وما يريده بالضبط من هذه الواجهة. ويقصد بالاحتياجات هنا هو ما تريده الجهة المالكة للنظام أو التي طلبت تصميم الواجهة, وتعتمد هذه الاحتياجات على مجموعة المهام والوظائف المطلوب من النظام أو التطبيق تنفيذها, حيث أن لكل نظام أو تطبيق مهام تختلف عن تلك التي يقوم بها نظام آخر. فالنظام الذي صُمِّمَ ليلبي احتياجات شركة هاتف نقال يختلف في وظائفه ومهامه عن التطبيق المستخدم كمحرك بحث على الانترنت وهذان يختلفان عن نظام الحجوزات المصمم لفندق سياحي وهكذا. وبالتالي فإن تصميم واجهة الاستخدام في كل حالة من الحالات التي ورد ذكرها يجب أن يراعي البيئة والظروف والمهام التي يجب على التطبيق القيام بها ليكون ذلك على أكمل وجه وفي أحسن صورة.




الشكل (2) مخطط تحديد المتطلبات

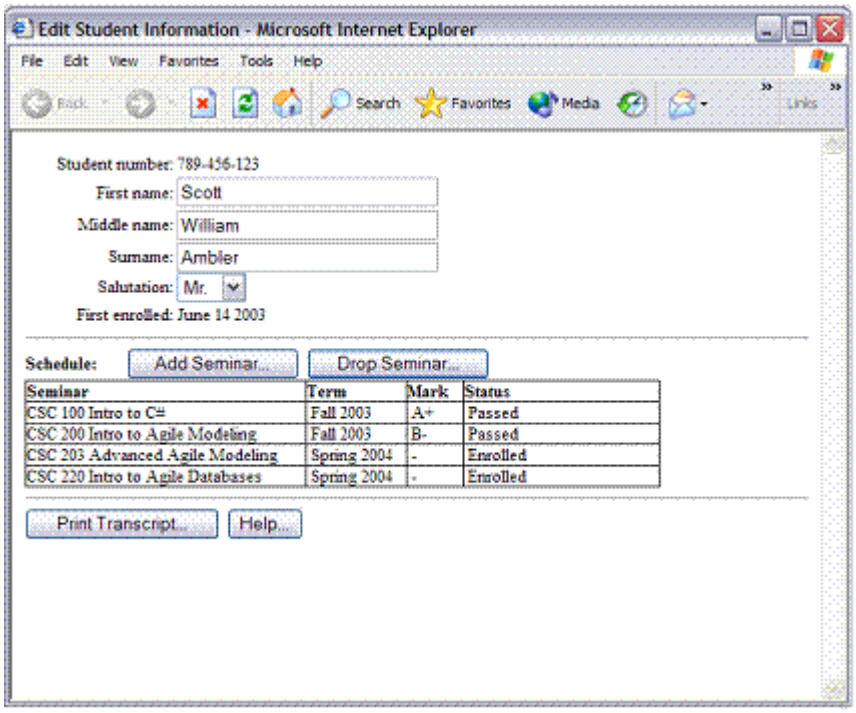
في الوقت الذي يتم فيه تحديد الاحتياجات يتم أيضا إنشاء ما يعرف بالنماذج الأولية لواجهة المستخدم (Essential User Interface Prototypes) والتي تكون على شكل مخططات ورسومات تجريبية أو مسودات (Sketches) تظهر عليها الملامح الأولوية والعناصر الأساسية للواجهة.

هنا ينتقل المصمم من مرحلة تعريف متطلبات المستخدم إلى مرحلة التحليل وهي النقطة التي يتم عندها اتخاذ قرار بتطوير جميع الأجزاء المكونة للنموذج الأولي أو بعضها فقط. هذا يعني أن الأفكار الأولية والملاحظات المبدئية التي كتبت بخط اليد وكذلك الرسومات والأشكال المتناثرة يتم تجميعها في نموذج أولي. تبدأ هذه العملية باتخاذ قرارات أساسية ومهمة تتحدد على أثرها معمارية الواجهة. على سبيل المثال تحديد هل ستكون الواجهة المزمع تصميمها هي لنظام واسع الانتشار كمتصفح الانترنت (Internet Browser) أم سيتم استخدامها كواجهة مستخدم رسومية GUI (Graphical User Interface) تعمل مع نظام Windows فقط. هذا التحديد سببه أن الأنواع المختلفة من التطبيقات يستخدم في تطويرها وبرمجتها لغات برمجة وأدوات برمجة مختلفة.




الشكل (3) مخطط نموذج أولي لواجهة مستخدم نظام قاعدة بيانات شؤون الطلبة

بعد تحديد الاحتياجات واعتماد النماذج الأولية والشكل النهائي الذي سوف تكون عليه الواجهة يتم الانتقال إلى المرحلة التالية وهي استخدام لغات البرمجة وأدواتها في تطوير الشاشات والصفحات والتقارير التي يحتاجها المستخدم عند تعامله مع التطبيق, حيث يجب اختيار لغة البرمجة المناسبة في تطوير الواجهة. على سبيل المثال في تطوير واجهات مواقع الانترنت تستخدم لغة HTML أما في تطوير واجهات نظام Windows فيتم استخدام لغة C.




**الشكل (4) واجهة تستخدم في مواقع الانترنت تم تطويرها باستخدام لغة HTML**

من المهم في هذه المرحلة الرجوع إلى الجهة صاحبة التطبيق لتقوم بتجربة الواجهة والتأكد من أن هذا التصميم بشكله الحالي يلبي احتياجات مستخدميها. عادة تقوم هذه الجهة بتقييم الواجهة من خلال تجربتها من قبل عدة مستخدمين ليتم في النهاية وضع إجابات على ثلاثة أسئلة مهمة هي:

1. ما هي الجوانب الإيجابية في التصميم ؟
2. ما هي الجوانب السلبية في التصميم ؟
3. ما هي الجوانب التي تم إغفالها ويجب أن يتضمنها التصميم ؟

بعد تقييم النموذج من المحتمل أن يتم حذف بعض الأجزاء أو المكونات من النموذج أو على العكس ربما تكون هناك حاجة لإضافة بعض المكونات والعناصر. ويتم إيقاف عملية التقييم عندما لا تُنْتِج تجربة الواجهة أية أفكار جديدة لتحسين عمل الواجهة أو أن تكون هذه الأفكار ليست ذات أهمية تذكر.

# تفاعل الإنسان والحاسوب

# Human-Computer Interaction

إن الكثير من الاختراعات والابتكارات التقنية يعود الفضل فيها إلى فعالية تصميم واجهة المستخدم (Efficacy of User Interface) حيث أن الكثير من الأنظمة والتطبيقات تكون على درجة كبيرة من التعقيد ولكنها تمتلك واجهات مستخدم على مستوى عالي من الكفاءة تجعل من استخدام هذه الأنظمة سهلا مما يعود بالفائدة القصوى على مستخدمي هذه الأنظمة. ففي الوقت الذي يركز فيه المهندسون على الجانب التقني لأي مُنْتَج يقوم مختصون بتصميم واجهات مستخدم بالبحث عن أفضل التصاميم التي تتيح للمستخدم الاستفادة القصوى من أمكانية هذا المنتَج. وللوصول إلى فعالية قصوى وسعر مناسب يقوم المهندسون ومصممي الواجهات بالتعاون مع بعضهم البعض من البداية إلى النهاية. ويمكن اعتبار تصميم واجهة المستخدم جيدا وناجحا إذا شعر المستخدم بأن الواجهة سهلة التعلم وبسيطة الاستخدام وتشعره بالراحة والرضا.

وللوصول إلى أفضل التصاميم لواجهات المستخدم يقوم متخصصون في علم تفاعل الإنسان والحاسوب HCI – وهو أحد علوم الحاسوب الحديثة نسبيا يهتم بتصميم وتقييم وتنفيذ نظم الحاسوب التفاعلية المعدة للاستخدام من قبل الإنسان وكذلك دراسة جميع الظواهر المحيطة بهذه الأنظمة - بدراسة كيفية استخدام الناس لأنظمة الحاسوب, ودراسة تأثير الحواسيب على الأفراد والمؤسسات والمجتمع , وتعمل هذه الدراسات على تسهيل استخدامهم للحاسوب عن طريق دعم المستخدمين وتحسين طريقة حصولهم على المعلومات وإنشاء أنظمة اتصالات أفضل وتشمل أيضا أدوات الإدخال والإخراج وتفاعل المستخدمين معها وكذلك الحصول على المعلومات ونشرها وتوثيق الملفات وأمور أخرى.

هذا العلم ليس منفصلا عن العلوم الأخرى بل هو عبارة عن تداخل مجموعة من العلوم مع بعضها البعض. فهو يحتاج إلى علم النفس وعلم الاجتماع مع علوم الحاسوب الأخرى لينجح, فدراسة احتياجات الناس وما يفضلونه يتدخل بها علم النفس وعلم الاجتماع بالإضافة إلى علوم الحاسوب المختلفة . أي أن هناك تداخل بين عدة علوم منها ما يتعلق بالسلوك الإنساني ومنها ما

يتعلق بعلوم الحاسوب. هذا التفاعل بين الإنسان والحاسوب يحدث عادة في واجهة المستخدم User Interface أو ببساطة الواجهة Interface التي تشمل البرمجيات والمعدات على حد سواء مثل طرفيات الحواسيب ذات الأغراض العامة والأنظمة الميكانيكية واسعة النطاق مثل الطائرات ومحطات توليد الطاقة.

هذا العلم يبحث في العلوم المتعلقة بالحاسوب مثل تقنيات الرسم بالحاسوب Computer Graphics, أنظمة التشغيل Operating Systems, لغات البرمجة Programming Languages وكذلك تطوير البيئة المحيطة بهذه الأنظمة هذا من ناحية, ومن ناحية أخرى فهو يهتم بدراسة العلوم الإنسانية مثل نظرية الاتصال Communication Theory, علم اللغويات Linguistics, علم الاجتماع Social Science, علم النفس الإدراكي Cognitive Psychology وغيرها. هذا التداخل له جانبان فمن جهة, هذا التعدد في العلوم التي تتم دراستها يغني هذه الدراسة ويفتح آفاق واسعة في التصميم والتطوير ولكنه في الوقت نفسه يجعل من الصعوبة بمكان تجميع هذه المعلومات واستخلاص المفيد منها.

يعتبر الهدف الأساسي من هذه الدراسة هو تحسين التفاعل بين المستخدمين والحواسيب وذلك بجعل هذه الحواسيب أكثر فاعلية وأكثر تقبلا لحاجات المستخدم. وبشكل محدد تهتم HCI بالأمور التالية:

1. أساليب وطرق تصميم واجهات المستخدم وذلك استنادا إلى مستوى المستخدم ونوع المهمات المطلوب تنفيذها حيث يتم اختيار أفضل تصميم للواجهة للوصول إلى اكبر قدر ممكن من الخصائص وكذلك إمكانية تعليم مهارات الاستخدام بفعالية.

2. طرق تنفيذ الواجهات (البرمجيات المستخدمة , المكتبات والخوارزميات ذات الكفاءة العالية).
3. تقنيات تقييم ومقارنة الواجهات.
4. تطوير واجهات جديدة وتطوير تقنيات التفاعل.

5. تطوير النماذج الوصفية والتنبؤية ونظريات التفاعل.

أما الهدف على المدى البعيد فهو تصميم أنظمة تقلل إلى أقصى حد الحواجز بين النموذج الإدراكي للإنسان الذي يريد انجاز مهمة معينة ومدى قابلية الحاسوب لتقبل هذه المهمات.

في بدايات ظهوره لم يكن الحاسوب يستخدم إلا في إجراء العمليات الحسابية وكان استخدامه مقتصرا على بعض المؤسسات العلمية والحكومية والشركات, ولكن دراسة HCI أسهمت بشكل كبير في تطور الحاسوب وتحسينه, فقد تم تقديم أفكار جديدة لواجهة المستخدم User Interface وأهمها التوصل إلى طريقة العرض من خلال النوافذ Windows وذلك باستخدام الواجهات الرسومية GUI (Graphical User Interface) التي تقوم في شكلها الحالي بعرض المعلومات بشكل واضح يسهل على المستخدمين –حتى الأطفال منهم- استخدام الحاسوب بحيث انه ليس من الضروري أن يكون الشخص متخصصا في الحاسوب لكي يتمكن من استخدام مصادر الحاسوب المادية والبرمجية على حد سواء بسهولة وبكفاءة عالية.

أهمية دراسة تفاعل الإنسان والحاسوب:

1. أظهرت الدراسات المختلفة أن واجهة المستخدم هي من أهم العوامل التي تؤدي إلى نجاح المنتج ورواجه بين الناس لذلك أصبح تركيز المبرمجين منصبا على تصميم وبرمجة واجهات مناسبة للاستخدام.
2. أدت هذه الدراسة إلى إنتاج أنظمة يسهل التعامل معها بعكس الأنظمة السابقة التي كانت تحتاج إلى خبرة واسعة في التعامل معها وهذا يبدو واضحا من الانتشار الواسع لأجهزة الحاسوب والهواتف النقالة والألعاب المختلفة التي تتميز جميعها بسهولة الاستخدام.
3. بفضل هذه الدراسة تم تطوير أنواع من الأنظمة والأدوات الجديدة كأدوات التعرف على الصوت والصورة Multimedia, كما تم تطوير شبكات الاتصالات العالمية وأنظمة نقل المعلومات التي لا تستغني عنها أي شركة حاسوب أو برمجيات.