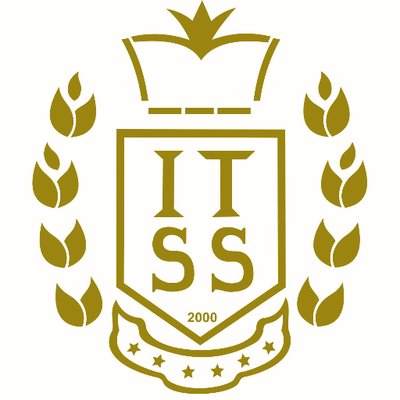
****

**INSTITUTO TECNOLÓGICO SUPERIOR DE LA REGIÓN SIERRA**

**INGENIERÍA INFORMÁTICA**

**SEMINARIO DE TIC**

**UNIDAD 4**

**MANYARI ROBLERO GIL - 16E30414**

**JHONATAN IVÁN GÓMEZ GARCÍA**

**M.I.A. ALEJANDRA GUADALUPE VÁZQUEZ MARTÍNEZ**

**TEAPA, TABASCO ABRIL/2020**

**INTRODUCCIÓN**

El desarrollo de aplicaciones conlleva un ciclo de vida con las siguientes fases: análisis, diseño, construcción, pruebas e implantación y soporte y mantenimiento. Para la realización de las mismas se cuenta con diferentes herramientas de apoyo.

Las aplicaciones se pueden [dividir en tres categorías](https://www.lancetalent.com/blog/tipos-de-aplicaciones-moviles-ventajas-inconvenientes/): las que atienden una necesidad específica, las que facilitan o agilizan una tarea, y aquellas que son de entretenimiento. Y cada una de ellas necesitará un plan de negocio diferente. Saber el tipo de aplicación que quieres desarrollar será tu punto de partida.

Para ello es importante hacerse algunas preguntas clave:

* ¿Qué necesidad/necesidades cubre tu aplicación?
* ¿Quién es tu audiencia?
* ¿Por qué le puede interesar tu aplicación?
* ¿Quiénes son y qué hacen tus competidores?
* ¿Cómo puedes destacar en el mercado?
* ¿Vas a desarrollar la App [para iOS o para Android](https://www.lancetalent.com/blog/crear-una-app-para-android-ios/)?

Es importante que te hagas estas preguntas para asegurarte de que tu idea vale la pena antes de llegar demasiado lejos en el proceso de desarrollo. En esta etapa debes crear también un [plan de negocios](https://www.lancetalent.com/blog/puntos-basicos-plan-de-negocio-pn/) y una [estrategia de monetización](https://www.lancetalent.com/blog/6-estrategias-infalibles-monetizar-app/) para tu App.

La **planificación** es clave para que cualquier proyecto se realice con éxito.

**UNIDAD 4: DESARROLLO DE APLICACIONES**

**4.1 CONCEPTO DE SISTEMA**

Un sistema informático. Puede ser definido como un sistema de información que basa la parte fundamental de su procesamiento, en el empleo de la [computación](https://www.ecured.cu/Computaci%C3%B3n), como cualquier [sistema](https://www.ecured.cu/Sistema), es un conjunto de funciones interrelacionadas, [hardware](https://www.ecured.cu/Hardware), [software](https://www.ecured.cu/Software) y de [Recurso Humano](https://www.ecured.cu/index.php?title=Recurso_Humano&action=edit&redlink=1). Un sistema informático normal emplea un sistema que usa dispositivos que se usan para programar y almacenar programas y datos.

Si además de la [información](https://www.ecured.cu/Informaci%C3%B3n), es capaz de almacenar y difundir los conocimientos que se generan sobre cierta temática, tanto dentro, como en el entorno de la entidad, entonces está en presencia de un [sistema de gestión de información y conocimientos](https://www.ecured.cu/index.php?title=Sistema_de_gesti%C3%B3n_de_informaci%C3%B3n_y_conocimientos&action=edit&redlink=1). Como utilizador final emplea esa información en dos actividades fundamentales: la toma de decisiones y el control.

Los sistemas informáticos son tipos de sistemas de información, o sea, sistemas que se organizan en torno al manejo de datos de diversa naturaleza, aunque no todos los sistemas de información sean informáticos. Esto es, **no todos son digitales, ni automatizados, ni electrónicos.**

Los sistemas informáticos ocupan en el mundo contemporáneo **un lugar clave para la organización humana**de sus procesos productivos y de otras naturalezas. Es una herramienta poderosa para el intercambio de información y la construcción de [redes informáticas](https://www.caracteristicas.co/redes-de-computadoras/) que superan la dificultad de las distancias.

**4.1.1 SISTEMAS DE CÓMPUTO**

Un sistema de cómputo se define como un conjunto de elementos organizados que interactúan  unos con otros para lograr ciertos objetivos operando sobre la información. Estos elementos son componentes físicos (hardware), los programas (software), los datos y los usuarios. Todos estos componentes son importantes y cada uno de ellos juega un papel fundamental para el correcto funcionamiento del sistema.

1. **HARDWARE.-**Son los componentes físicos y tangibles es decir, las partes que se pueden ver y tocar. Incluye la torre de control, dispositivos de entrada, almacenamiento y salida
2. **SOFTWARE.-** Son los componentes lógicos e intangibles; son los programas que le indican al hardware las tareas que debe ejecutar. Estos programas permiten operar la computadora, configurar los dispositivos y realizar aplicaciones específicas, como el procesador de textos.

Los programas o software son elementos intangibles o lógicos que posibilitan  que la computadora realice todos los procesos, transformándola la herramienta por excelencia del siglo XXI para los negocios, las comunicaciones y en general para casi cualquier actividad del ser humano.

El **software es una parte indispensable para el funcionamiento de la computadora. Está formado por un conjunto de instrucciones y datos que nos permiten aprovechar las capacidades que tiene la computadora para ayudarnos en la solución de problemas.**  
De esta manera, el software tiene diferentes funciones:

* Administrar los recursos o medios de la computadora.
* Proporciona herramientas para usar eficientemente estos recursos.
* Actúa como enlace entre el usuario y la computadora.

**4.1.2 SISTEMAS DE INFORMACIÓN**

Conjunto de elementos que interactúan entre sí con un fin común; que permite que la información esté disponible para satisfacer las necesidades en una organización, un sistema de información no siempre requiere contar con recuso computacional aunque la disposición del mismo facilita el manejo e interpretación de la información por los usuarios.  
  
 Los elementos que interactúan entre sí son: el equipo computacional (cuando este disponible), el recurso humano, los datos o información fuente, programas ejecutados por las computadoras, las telecomunicaciones y los procedimientos de políticas y reglas de operación.  
  
Un Sistema de Información realiza cuatro actividades básicas:

* Entrada de información: proceso en el cual el sistema toma los datos que requiere.
* Almacenamiento de información: pude hacerse por computadora o archivos físicos para conservar la información.
* Procesamiento de la información: permite la transformación de los datos fuente en información que puede ser utilizada para la toma de decisiones
* Salida de información: es la capacidad del sistema para producir  la información procesada o  sacar  los datos de entrada al exterior.

Los usuarios de los sistemas de información tienen diferente grado de participación dentro de un sistema y son el elemento principal que lo integra, así se puede definir usuarios primarios quienes alimentan el sistema, usuarios indirectos que se benefician de los  resultados pero que no interactúan con el sistema,  usuarios gerenciales y directivos quienes tienen responsabilidad administrativa y de toma de decisiones con base a la información que produce el sistema.

**4.2 METODOLOGÍAS**

*METODOLOGÍA TRADICIONAL*

Al inicio el desarrollo de software era artesanal en su totalidad, la fuerte necesidad de mejorar el proceso y llevar los proyectos a la meta deseada, tuvieron que importarse la concepción y fundamentos de metodologías  existentes en otras áreas y adaptarlas al desarrollo de software. Esta nueva etapa de adaptación contenía el desarrollo dividido en etapas de manera secuencial que de algo mejoraba la necesidad latente en el campo del software.

Entre las principales metodologías tradicionales tenemos los ya tan conocidos RUP y MSF entre otros, que centran su atención en llevar una documentación exhaustiva de todo el proyecto y centran su atención en cumplir con un plan de proyecto, definido todo esto, en la fase inicial del desarrollo del proyecto.

Otra de las características importantes dentro de este enfoque tenemos los altos costos al implementar un cambio y al no ofrecer una buena solución para proyectos donde el  entorno es volátil.

Las metodologías tradicionales (formales) se focalizan en documentación, planificación y procesos. (Plantillas, técnicas de administración, revisiones, etc.), a continuación se detalla RUP uno de los métodos más usados dentro de los métodos tradicionales.

**SCRUM**

* Provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo.
* Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales (respetando cronograma y presupuesto).
* Fue desarrollado por Rational Software.
* Tiene un conjunto de reglas muy pequeño y muy simple y está basado en los principios de inspección continua, adaptación, auto-gestión e innovación.

**ICONIX:** *MICROSOFT SOLUTION FRAMEWORK (MSF)*

* El modelo de equipos de MSF tiene seis roles que corresponden a las metas principales de un proyecto y son responsables por las mismas.
* Cada rol puede estar compuestos por una o más personas.
* La estructura circular del modelo, muestra que no es un modelo jerárquico y que cada todos los roles son igualmente importantes en su aporte al proyecto.
* Maneja casos de uso, como el RUP, pero le falta mucho para llegar al nivel del RUP.
* El equipo implanta la tecnología base y los componentes relacionados.
* Estabiliza la instalación, traspasa el proyecto al personal de soporte y operaciones.
* Obtiene la aprobación final del cliente.

**Modelo de cascada**

Si alguna vez has incursionado en el mundo del [Desarrollo de Software](https://www.megapractical.com/blog-de-procesos-de-negocios-arquitectura-soa-desarrollo-de-software-business-intelligence/que-es-cmmi-y-por-que-es-importante-para-el-desarrollo-de-software), de seguro te has topado en algún momento con el modelo de cascada. De no ser así, cabe destacar que en este modelo cada etapa representa una unidad de desarrollo con un pequeño descanso en el medio. Por lo tanto, cada siguiente etapa inicia tan pronto como la anterior haya culminado, y esos descansos son usados para confirmaciones del lado del cliente.

Adicionalmente, este es considerado como el método tradicional de explicar el proceso de desarrollo de software en ingeniería de software, por lo que actualmente es visto como anticuado. Sin embargo, aún sigue siendo aplicado a proyectos con metas claras y requisitos que demandan hasta 100 horas de desarrollo, sobre todo considerando que este enfoque permite a los negocios deshacerse del papeleo innecesario, reuniones regulares que consumen mucho tiempo y retrasos en sus procesos de negocio.

Es por esto que esta es una gran opción para pequeños proyectos donde todos los aspectos del [proceso de desarrollo de software](https://www.megapractical.com/desarrollo-de-software-en-mexico-ecuador-y-panama-leg-1) se conocen de antemano, pero una mala solución para proyectos complicados, ya que se trata de un modelo bastante inflexible.

**Modelo de Espiral**

Mientras que la metodología de la cascada ofrece una estructura ordenada para el desarrollo de software, las demandas de tiempo reducido al mercado hacen que sus pasos en serie sean inapropiados.

El siguiente paso evolutivo desde la cascada es donde se realizan los diversos pasos para múltiples entregas o traspasos. La última evolución de la caída del agua es la espiral, aprovechando el hecho de que los proyectos de desarrollo funcionan mejor cuando son incrementales e iterativos. La metodología espiral refleja la relación de tareas con prototipos rápidos, mayor paralelismo y concurrencia en las actividades de diseño y construcción. El método en espiral debe todavía ser planificado metódicamente, con las tareas y entregables identificados para cada paso en la espiral.

*METODOLOGÍA ÁGIL*

Son aquellas que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno.

En esencia, las empresas que apuestan por esta metodología consiguen gestionar sus proyectos de forma flexible, autónoma y eficaz reduciendo los costes e incrementando su productividad.

**Metodología de Programación Extrema (XP)**

Como metodología ágil de ingeniería de software, la metodología de programación extrema se conoce actualmente como metodología de XP (eXtreme Programming). Esta metodología, se utiliza principalmente para evitar el desarrollo de funciones que actualmente no se necesitan, pero sobre todo para  para atender proyectos complicados. Sin embargo, sus métodos peculiares pueden tomar más tiempo, así como recursos humanos en comparación con otros enfoques.

Estas son solo algunas de las metodologías de Desarrollo de Software que existen, pero lo importante es que tengas en cuenta que al estar familiarizado con estos populares enfoques podrás optimizar la eficiencia de tus proyectos utilizando un enfoque puro o combinando algunos de ellos.

**Kanban**

La estrategia [Kanban](https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/) conocida como ‘Tarjeta Visual» muy útil para los responsables de proyectos. Esta consiste en la elaboración de un cuadro o diagrama en el que se reflejan tres columnas de tareas; pendientes, en proceso o terminadas. Este cuadro debe estar al alcance de todos los miembros del equipo, evitando así la repetición de tareas o la posibilidad de que se olvide alguna de ellas. Por tanto, ayuda a mejorar la productividad y eficiencia del equipo de trabajo.

**Metodología de Prototipo**

Es un procedimiento de desarrollo especializado que permite a los desarrolladores la posibilidad de poder solo hacer la muestra de la resolución para poder  validar su esencia funcional ante los clientes, y hacer los cambios que sean fundamentales antes de crear la solución final auténtica. De hecho, la mejor parte de esta metodología es que tiende a resolver un conjunto de problemas de diversificación que ocurren con el método de la cascada.

Además de esto, la gran ventaja de optar por este enfoque es que da una idea clara sobre el proceso funcional del software, reduce el riesgo de falla en una funcionalidad de software y asiste bien en la recolección de requisitos y en el análisis general.

**Desarrollo Rápido de Aplicaciones (RAD)**

Con el objetivo de otorgar resultados rápidos, se trata de un enfoque que está destinado a proporcionar un excelente procesos de desarrollo con la ayuda de otros enfoques, pero además, está diseñado para aumentar la viabilidad de todo el procedimiento de desarrollo de software para resaltar la participación de un usuario activo.

Dicho esto, algunas de las ventajas a destacar de este tipo de desarrollo son las siguientes:

* Hace todo el proceso de desarrollo sin esfuerzo.
* Asiste al cliente en la realización de revisiones rápidas.
* Alienta la retroalimentación de los clientes para su mejora.

*METODOLOGÍA HIBRIDA*

Recogen las bondades y beneficios de las metodologías tradicional y ágil, pudiendo de esta manera adaptarse a las necesidades de cada organismo.

El uso de estás metodologías se ha ido incrementando de manera paulatina, ya que se adaptan a la cultura y necesidades de cada organismo, generando un incremento de productividad. Algunos ejemplos de este tipo de metodología son: EssUP (Essencial Unified Process), la combinación de Cascada y Scrum y SXP (Scrum y XP).

Entre los beneficios del uso de la metodología híbrida, podemos encontrar:

* Simple y fácil de entender y usar.
* Mejora continua exponencial.
* Flexibilidad a cambios dentro de un alcance de presupuesto definido.

**4.3 MODELOS DE PROGRAMACIÓN**

Los tipos o técnicas de programación son bastante variados, aunque puede que muchos de los lectores sólo conozcan una metodología para realizar programas. En la mayoría de los casos, las técnicas se centran en programación modular y programación estructurada, pero existen otros tipos de programación. Los explicaremos a lo largo del artículo.  
  
**Programación estructurada (PE)**

La programación estructurada esta compuesta por un conjunto de técnicas que han ido evolucionando aumentando considerablemente la productividad del programa reduciendo el tiempo de depuración y mantenimiento del mismo.  
Esta programación estructurada utiliza un número limitado de estructuras de control, reduciendo así considerablemente los errores.

Esta técnica incorpora:

* Diseño descendente (top-dow): el problema se descompone en etapas o estructuras jerárquicas.
* Recursos abstractos (simplicidad): consiste en descompones las acciones complejas en otras más simples capaces de ser resueltas con mayor facilidad.
* Estructuras básicas: existen tres tipos de estructuras básicas:
* Estructuras secuénciales: cada acción sigue a otra acción secuencialmente. La salida de una acción es la entrada de otra.
* Estructuras selectivas: en estas estructuras se evalúan las condiciones y en función del resultado de las mismas se realizan unas acciones u otras. Se utilizan expresiones lógicas.
* Estructuras repetitivas: son secuencias de instrucciones que se repiten un número determinado de veces.

Las principales ventajas de la programación estructurada son:

* Los programas son más fáciles de entender
* Se reduce la complejidad de las pruebas
* Aumenta la productividad del programador
* Los programas queden mejor documentados internamente.

Un programa está estructurado si posee un único punto de entrada y sólo uno de salida, existen de "1 a n" caminos desde el principio hasta el fin del programa y por último, que todas las instrucciones son ejecutables sin que aparezcan bucles infinitos.  
  
**Programación modular**

En la programación modular consta de varias secciones dividas de forma que interactúan a través de llamadas a procedimientos, que integran el programa en su totalidad. En la programación modular, el programa principal coordina las llamadas a los módulos secundarios y pasa los datos necesarios en forma de parámetros. A su vez cada módulo puede contener sus propios datos y llamar a otros módulos o funciones.

**Programación orientada a objetos (POO)**

Se trata de una técnica que aumenta considerablemente la velocidad de desarrollo de los programas gracias a la reutilización de los objetos. El elemento principal de la programación orientada a objetos es el objeto. El objeto es un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.  
 Un objeto contiene varios datos bien estructurados y pueden ser visibles o no dependiendo del programador y las acciones del programa en ese momento. El polimorfismo y la herencia son unas de sus principales características y por ello dedicaremos más adelante un artículo exclusivamente a tratar estos dos términos.

**Programación concurrente**

Este tipo de programación se utiliza cuando tenemos que realizar varias acciones a la vez. Se suele utilizar para controlar los accesos de usuarios y programas a un recurso de forma simultánea. Se trata de una programación más lenta y laboriosa, obteniendo unos resultados lentos en las acciones.

**Programación funcional**

Se caracteriza principalmente por permitir declarar y llamar a funciones dentro de otras funciones.

**Programación lógica**

Se suele utilizar en la inteligencia artificial y pequeños programas infantiles. Se trata de una programación basada en el cálculo de predicados (una teoría matemática que permite lograr que un ordenador basándose en hecho y reglas lógicas, pueda dar soluciones inteligentes).

**4.4 ESTRATEGIAS DE DISEÑO DE BASE DE DATOS**

* **Determinar el propósito de la base de datos**; Esto le ayuda a prepararse para el resto de los pasos.
* **Buscar y organizar la información necesaria**: Recopile todos los tipos de información que desea grabar en la base de datos, como el número de nombre y orden de producto.
* **Dividir la información en tablas**: Dividir los elementos de información en entidades o temas, como productos o pedidos principales. Cada tema se convierte en una tabla.
* **Convertir los elementos de información en columnas**: Decida qué información desea almacenar en cada tabla. Cada elemento se convierte en un campo y se muestra como una columna de la tabla. Por ejemplo, una tabla empleados puede incluir campos como apellido y fecha de contratación.
* **Especificar claves principales**: Elija la clave principal de cada tabla. La clave principal es una columna que se utiliza para identificar inequívocamente cada fila. Un ejemplo podría ser el identificador de producto o Id.
* **Configurar las relaciones de tabla**: Examine cada tabla y decida cómo los datos en una tabla relacionados con los datos en otras tablas. Agregar campos a las tablas o cree nuevas tablas para clarificar las relaciones según sea necesario.
* **Refinar el diseño**: Analizar el diseño de errores. Crear las tablas y agregue algunos registros de datos de ejemplo. Vea si puede obtener los resultados que desee en las tablas. Realizar ajustes en el diseño, según sea necesario.
* **Aplicar las reglas de normalización**: Aplicar las reglas de normalización de datos para ver si las tablas están estructuradas correctamente. Realizar ajustes en las tablas, según sea necesario.
  + **Etapa del diseño conceptual**: en esta etapa se obtiene una estructura de la información de la futura base de datos independiente de la tecnología que se empleará. No se tiene en cuenta todavía qué tipo de base de datos se utilizará (relacional, orientada a objetos, jerárquica); tampoco se tiene en cuenta con qué SGBD (sistema de gestión de base de datos) ni con qué lenguaje concreto se implementará la base de datos. El resultado de esta etapa es un modelo de flujo de información de alto nivel, uno de los más empleados es el modelo entidad relación (ER) y se obtiene luego de entrevistas, visitas y una investigación adecuada del sistema de información. El diagrama de entidad relación utiliza formas para representar entidades, atributos y relaciones.
  + **Etapa del diseño lógico:** en esta etapa se parte del resultado del diseño conceptual, que se transforma al tipo de base de datos que vamos a utilizar. Más concretamente, es preciso que se ajuste al modelo del SGBD con el que se desea implementar la base de datos. Por ejemplo, si se trata de un SGBD relacional, esta etapa obtendrá un conjunto de relaciones donde las entidades se transforman a tablas normalizadas con sus atributos, claves primarias y claves foráneas. El proceso de normalización que se aplica en esta etapa consiste en una serie de reglas que deben cumplir las tablas y relaciones obtenidas tras el paso del modelo entidad relación al modelo relacional, para entonces ser un modelo lógico. Las bases de datos relacionales se normalizan básicamente para: evitar la redundancia de los datos, evitar problemas de actualización de los datos en las tablas, proteger la integridad de los datos.
  + **Etapa del diseño físico:** en esta etapa se transforma la estructura obtenida en la etapa del diseño lógico, con el objetivo de conseguir una mayor eficiencia; además, se completa con aspectos de implementación física que dependerán del SGBD.

**4.5 HERRAMIENTAS CASE**

* **HERRAMIENTAS DE LA INGENIERÍA DE LA INFORMACIÓN:** Estas herramientas CASE modelan la información de negocios cuando ésta se transfiere entre distintas entidades organizativas en el seno de una compañía. El objetivo primordial de las herramientas de esta categoría consiste en representar objetos de datos de negocios, sus relaciones, y ayuda a comprender mejor la forma en que fluyen estos objetos de datos entre distintas zonas de negocio en el seno de la compañía. Estas herramientas proporcionan una ayuda importante cuando se diseñan nuevas estrategias para los sistemas de información y cuando los métodos y sistemas no satisfacen las necesidades de la organización.
* **MODELADO DE PROCESOS Y HERRAMIENTAS DE ADMINISTRACIÓN:** Se utilizan para representar los elementos clave del proceso de modo que sea posible entenderlo mejor. Estas herramientas también pueden proporcionar vínculos con descripciones de procesos que ayuden a quienes estén implicados en el proceso de comprender las tareas que se requieren para llevar a cabo ese proceso. Las herramientas de administración de procesos pueden proporcionar vínculos con otras herramientas que proporcionen un apoyo para actividades de proceso ya definidas.
* **HERRAMIENTAS DE PLANIFICACIÓN DE PROYECTOS.**

Las herramientas de esta categoría se concentran en dos áreas primordiales:

* *Estimación de esfuerzos de proyecto y de costes de software*. Calculan el esfuerzo estimado, la duración del proyecto y el numero recomendado de personas.
* *Planificación de proyectos*. Capacitan al administrador para definir todas las áreas del proyecto (la estructura de desglose de tareas), para crear una red de tareas (normalmente empleando una entrada gráfica), para representar las interdependencias entre tareas y para modelar la cantidad de paralelismo que sea posible para ese proyecto.
* **HERRAMIENTAS DE ANÁLISIS DE RIESGOS:** Las herramientas de análisis de riesgos capacitan al administrador el proyecto para construir una tabla de riesgos proporcionando una guía detallada en la identificación y análisis de riesgos.
* **HERRAMIENTAS DE ADMINISTRACIÓN DE PROYECTOS:** La planificación del proyecto y el plan del proyecto deben seguirse y de monitorizarse de forma continua. Además, el gestor deberá de utilizar las herramientas que recojan métricas que en la ultima instancia proporcionen una indicación de la calidad el producto del software. Las herramientas de esta categoría suelen ser extensiones de herramientas de planificación de proyectos.
* **HERRAMIENTAS DE SEGUIMIENTO DE REQUISISTOS:** Cuando se desarrollan grandes sistemas, el sistema proporcionado suele no satisfacer los requisitos especificados por el cliente. El objetivo de las herramientas de seguimiento de requisitos es proporcionar un enfoque sistemático para el aislamiento de requisitos, comenzando por las especificaciones del cliente. Las herramientas de trazado de requisitos típicos combinan una evaluación de textos por interacción humana, con un sistema de gestión de bases de datos que almacena y categoría todos y cada uno de los requisitos del sistema que se "analizan" a partir de las especificaciones originales.
* **HERRAMIENTAS DE MÉTRICAS Y GESTIÓN:** Las métricas del software mejoran la capacidad del administrador para controlar y coordinar el proceso del software y la capacidad del ingeniero para mejorar la calidad del software que se produce.

Las herramientas métricas actuales se centran en procesos, proyectos y características del producto.

Las herramientas orientadas a la gestión capturan métricas especificas del proyecto (por ejemplo: LDC/personamos, defectos por punto de función) que proporcionan una indicación global de productividad o de calidad. Las herramientas orientadas técnicamente determinan métricas técnicas que proporcionan una mejor visión de la calidad del diseño o del código. Muchas de las herramientas métricas avanzadas mantiene una base de datos de medidas de medias de la industria.

Basándose en características de proyectos y de productos proporcionados por el usuario, estas herramientas califican los números locales frente a los valore medios de la industria (y frente al rendimiento local anterior) y sugieren estrategias para llegar a mejoras. Estas herramientas utilizan un sistema experto para sugerir el orden en el que se debe llevar a cabo un proyecto.

* **HERRAMIENTAS DE DOCUMENTACIÓN**: Las herramientas de producción de documentos y autoedición prestan su apoyo a casi todos los aspectos de la ingeniería del software, y representan una importante oportunidad de aprovechamiento para todos los desarrolladores del software. La mayor parte de las organizaciones dedicadas al desarrollo de software invierte una cantidad de tiempo considerable en el desarrollo de documentos, y en muchos casos el proceso de documentación en si resulta bastante deficiente. No es raro que una organización de desarrollo de software invierta hasta en un 20 o 30 pro ciento de su esfuerzo global de desarrollo de software en la documentación. Por esta razón, las herramientas de documentación suponen una oportunidad importante para mejorar la productividad.
* **HERRAMIENTAS DE SOFTWARE DE SISTEMA:** CASE es una tecnología de estaciones de trabajo. Por tanto, el entorno CASE debe adaptase a un software de sistema en redes de alta calidad, al correo electrónico, a los boletines electrónicos y a otras capacidades de comunicaciones.
* **HERRAMIENTAS DE CONTROL DE CALIDAD:** La mayor parte de las herramientas CASE que afirman que tiene como principal interés el control de calidad son en realidad herramientas métricas que hace una auditoria del código fuente para determinar si es justa o no a ciertos estándares del lenguaje. Otras herramientas extraen métricas técnicas como base para medir la calidad del software que se esta construyendo.
* **HERRAMIENTAS DE GESTIÓN COMO BASE DE DATOS:** El software de gestión de bases de datos sirve como fundamentos para establecer una base de datos CASE. Dado el énfasis acerca de los objetos de configuración, las herramientas de gestión de bases de datos para CASE pueden evolucionar a partir de los sistemas de gestión de bases de datos relacionales (SGBDR) para transformarse en sistemas de gestión de bases de datos orientadas a objetos(SGBDOO).
* **HERRAMIENTAS DE CODIFICACIÓN DE CUARTA GENERACIÓN:** Los sistemas de consulta de bases de datos, los generadores de código y los lenguajes de cuarta generación han cambiado la forma en que se desarrollan los sistemas. Idealmente, estas herramientas de generación de código no solo traducen la descripción de un sistema operativo, sino que también ayudan a verificar la corrección de la especificación del sistemas de tal forma que la salida resultante satisfaga los requisitos del usuario.

Los lenguajes de cuarta generación se usan ampliamente en aplicaciones de sistemas de información.

Aunque los lenguajes de cuarta generación, los generadores de código y los generadores de aplicaciones, permiten que un ingeniero de software especifique un sistema a un nivel muy alto de abstracción; cada una de estas herramientas difiere en aspectos importantes.

* **HERRAMIENTAS DE MANTENIMIENTO**: Las herramientas CASE para el mantenimiento de software abarcan una actividad que actualmente ocupa, aproximadamente, el 70% del esfuerzo total dedicado al software. La categoría de herramientas de mantenimiento puede subdividirse de la siguiente forma:
* *Herramientas de ingeniería inversa a especificaciones*. Toman el código fuente como entrada y generan modelos de diseño y análisis estructurado, listas de utilización y otra información con el diseño.
* *Herramientas de reestructuración y análisis de código*. Analizan la sintaxis del programa, generan un grafo de flujo de control y un programa estructurado.
* *Herramientas interactivas de reingeniería de sistema*. Se utilizan para modificar sistemas de base de datos.

Estas herramientas están limitadas a lenguajes de programación específicos y requieren cierto grado de interacción con el ingeniero de software.

* **HERRAMIENTAS DE GESTIÓN DE CONFIGURACIÓN DE SOFTWARE:** La gestión de configuración de software (GCS) se encuentra en el núcleo de todos los entornos CASE. Las herramientas pueden ofrecer su asistencia en las cinco tareas principales de GCS: identificación, control de versiones control de cambios, auditoria y contabilidad de estados. La base de datos CASE proporciona un mecanismo para identificar todos los elementos de configuración y relacionarlo con otros elementos; un acceso sencillo a los elementos de configuración individuales facilita el proceso de auditoria; las herramientas de comunicación CASE pueden mejorar enormemente la contabilidad de estados (ofreciendo información acerca de los cambios a todos aquellos que necesiten conocerlos).
* **HERRAMIENTAS DE ANÁLISIS Y DISEÑO\_** Las herramientas de análisis y diseño capacitan al ingeniero del software para crear modelos del sistema que haya que construir. Los modelos contienen una representación de los datos, de la función y del comportamiento (en el nivel de análisis), así como caracterizaciones del diseño de datos, arquitectura, procedimientos e interfaz. Al efectuar una comprobación de la consistencia y validez del modelo, las herramientas de análisis y diseño proporcionan al ingeniero del software un cierto grado de visión en lo tocante a la representación del análisis, y le ayudan a eliminar errores antes de que se propaguen al diseño, o lo que es peor, a la propia implementación.
* **HERRAMIENTAS PRO/SIM:** Las herramientas PRO/SIM (de prototipos y simulación) proporcionan al ingeniero del software la capacidad de predecir el comportamiento de un sistema en tiempo real antes de llegar a construirlo. Además, capacitan al ingeniero del software para desarrollar simulaciones del sistema de tiempo real que permitirán al cliente obtener ideas acerca de su funcionamiento, comportamiento y respuesta antes de la verdadera implementación.
* **HERRAMIENTAS DE DESARROLLO Y DISEÑO DE INTERFAZ:** Las herramientas de desarrollo y diseño de interfaz son en realidad un conjunto de primitivas de componente de programas tales como menús, botones, estructuras de ventanas, iconos, mecanismos de desplazamiento, controladores de dispositivos, etc., Sin embargo, estos conjuntos de herramientas se están viendo sustituidos por herramientas de generación de prototipos de interfaz que permiten una rápida creación en pantalla de sofisticadas interfaces de usuario, que se ajustan al estándar de interfaz que se haya adoptado para el software.
* **HERRAMIENTAS DE GENERACIÓN DE PROTOTIPOS:** Se puede utilizar toda una gama de herramientas de generación de prototipos. Los generadores de pantallas permiten al ingeniero de software definir rápidamente la disposición de pantalla para aplicaciones interactivas. Otras herramientas de prototipos CASE mas sofisticadas permiten la creación de un diseño de datos, acoplado con las disposiciones de la pantalla y de los informes simultáneamente. Muchas herramientas de análisis y diseño proporcionan extensiones que ofrecen alguna opción de generación de prototipos. Las herramientas PRO/SIM generan un esqueleto de código fuente en Ada y C para las aplicaciones de ingeniería (en tiempo real). Por ultimo, una gama de herramientas de cuarta generación poseen también características de generación de prototipos.
* **HERRAMIENTAS DE PROGRAMACIÓN:** La categoría de herramientas de programación abarca los compiladores, editores y depuradores que están disponibles para prestar su apoyo en la mayoría de los lenguajes de programación convencionales. Además, los entornos de programación orientados a objetos (OO), los lenguajes de cuarta generación, los entornos de programación gráfica, los generadores de aplicaciones y los lenguajes de consulta de bases de datos residen también en esta categoría.
* **HERRAMIENTAS DE INTEGRACIÓN Y COMPROBACIÓN\_** En su directorio de herramientas de comprobación de software, software Quality Engineering define las siguientes categorías de herramientas de comprobación:
* *Adquisición de datos:* herramientas que adquieren datos que se utilizaran durante la comprobación.
* *Medida estática:* herramientas que analizan el código fuente sin ejecutar casos de prueba.
* *Medida dinámica:* herramientas que analizan el código fuente durante la ejecución.
* *Simulación:* herramientas que simulan las funciones del hardware o de otros elementos externos.
* *Administración de comprobaciones:* herramientas que prestan su asistencia en la planificación, desarrollo y control de las comprobaciones.
* *Herramientas de funcionalidad cruzada:* se trata de herramientas que cruzan los límites de las categorías anteriores.

Debería tenerse en cuenta que muchas de las herramientas de comprobación poseen características que abarcan dos o más de las categorías anteriores.

* **HERRAMIENTAS DE ANÁLISIS ESTÁTICO:** Las herramientas de análisis estático prestan su asistencia al ingeniero del software a efectos de derivar casos prácticos. Se utilizan tres tipos distintos de herramientas estáticas de comprobación en la industria: herramientas de comprobación basadas en código, lenguajes de comprobación especializados, y herramientas de comprobación basadas en requisitos. Las herramientas de comprobación basadas en código admiten un código fuente (o PDL) como entrada y efectúan un cierto numero de análisis que can lugar a la generación de casos de prueba. Los lenguajes de comprobación especializados (por ejemplo: ATLAS) capacitan al ingeniero del software para escribir detalladas especificaciones de comprobación que describirán todos los casos de prueba y la logística de su ejecución. Las herramientas de comprobación basadas en requisitos aíslan requisitos específicos del usuario y sugieren casos de prueba (o clases de comprobaciones) que ejerciten estos requisitos.
* **HERRAMIENTAS DE ANALISIS DINAMICO:** Las herramientas de análisis dinámico interactúan con un programa que se esté ejecutando, comprueban la cobertura de rutas, comprueban las afirmaciones acerca del valor de variables especificas y en general instrumentan el flujo de ejecución del programa. Las herramientas dinámicas pueden ser bien intrusivas, bien no intrusivas. Las herramientas intrusivas modifican el software que hay que comprobar mediante sondas que se insertan (instrucciones adicionales) y que efectúan las actividades mencionadas anteriormente. Las herramientas de comprobación no intrusivas utilizan un procesador hardware por separado que funciona en paralelo con el procesador que contenga el programa que se está comprobando.
* **HERRAMIENTAS DE GESTIÓN DE COMPROBACIÓN:** Las herramientas de gestión de comprobación se utilizan para comprobar y coordinar la comprobación de software para cada uno de los pasos principales de comprobación. Las herramientas de esta categoría administran y coordinan la comprobación de regresiones, efectúan comparaciones que determinan las diferencia s entre la salida real y la esperada, y efectúan comprobaciones por lotes de programas con interfaces interactivas entre hombre y maquina. Además de las funciones indicadas anteriormente, muchas herramientas de gestión de comprobaciones sirven también como controladores de comprobación genéricos. Un controlador de comprobación lee uno o mas casos de prueba de algún archivo de pruebas, da formato a los datos de prueba para que se ajusten a las necesidades del software que se esta probando, e invoca entonces al software que sea preciso comprobar.
* **HERRAMIENTAS DE COMPROBACIÓN CLIENTES/SERVIDOR:** El entorno C/S existe unas herramientas de comprobación especializadas que ejerciten la interfaz gráfica de usuario y los requisitos de comunicaciones en red par el cliente y el servidor.
* **HERRAMIENTAS DE REINGENIERIA;** La categoría de herramientas de reingeniería se pueden subdividir en las funciones siguientes:
* *Herramientas de ingeniería inversa para producir especificaciones*: se toma el código fuente como entrada y se generan modelos gráficos de análisis y diseño estructurados, listas de utilización y otras informaciones de diseño.
* *Herramientas de reestructuración y análisis de código*: se analiza la sintaxis del programa, se genera una gráfica de control de flujo y se genera automáticamente un programa estructurado.
* *Herramientas de reingeniería para sistemas en línea*: se utilizan para modificar sistemas de bases de datos en línea (por ejemplo: para convertir archivos IDMS o DB2 traduciéndolos a un formato de entidades y relaciones).

Muchas de las herramientas anteriores están limitadas a lenguajes de programación específicos (aun cuando se abarcan la mayoría de los lenguajes principales) y requieren un cierto grado de interacción con un ingeniero del software.

Las herramientas de ingeniería inversa y progresiva de la próxima generación harán un uso mucho mayor de técnicas de inteligencia artificial, aplicando una base de conocimientos que se a especifica del dominio de la aplicación (esto es, un conjunto de reglas de descomposición que se aplicarían a todos los programas de una cierta zona de aplicación tal como el control de fabricación o la aviónica). El componente de inteligencia artificial asistirá en la descomposición y reconstrucción del sistemas, pero seguirá requiriendo una interacción con un ingeniero de software a lo largo del ciclo de la reingeniería.

**CONCLUSIÓN**

Las herramientas ayudan a averiguar cómo los usuarios van a interactuar con la app. Es decir, cómo los botones y menús dirigen a las diferentes pantallas de la aplicación. La interfaz de usuario determinará la apariencia y los elementos de diseño de tu app.

Una vez que tengas clara la idea puedes empezar a [buscar un desarrollador](https://lancetalent.com/index.php/app/)**para tu app** No tengas miedo de externalizar el desarrollo.

Después del lanzamiento debes evaluar el éxito de la aplicación para determinar quiénes son tus usuarios y por qué eligieron tu producto. Si no es la audiencia que esperabas es hora de ajustar tus esfuerzos de marketing.

Puedes incluir nuevas funciones y características en la próxima versión de tu aplicación o cambiar ciertos aspectos del diseño para mejorar la experiencia del usuario. El día de lanzamiento es sólo el principio, y si permaneces atento y abierto a nuevas ideas, tu aplicación puede lograr un éxito continuo.

**REFERENCIAS ELECTRÓNICAS**

**Recuperado por Manyari Roblero Gil el 27 de abril de 2020:** <https://www.caracteristicas.co/sistema-informatico/#ixzz6KfbCSezy>

**Recuperado por Jhonatan Gómez García el 27 de abril de 2020:** <https://www.caracteristicas.co/sistema-informatico/>

**Recuperado por Manyari Roblero Gil el 29 de abril de 2020:** <http://eninformatic.blogspot.com/2012/08/sistema-de-computo_27.html>

**Recuperado por Jhonatan Gómez García el 27 de abril de 2020:** <http://www.incap.int/sisvan/index.php/es/acerca-de-san/conceptos/797-sin-categoria/501-sistema-de-informacion>

**Recuperado por Manyari Roblero Gil el 29 de abril de 2020:** <https://www.megapractical.com/blog-de-arquitectura-soa-y-desarrollo-de-software/metodologias-de-desarrollo-de-software>

**Recuperado por Jhonatan Gómez García el 27 de abril de 2020:** <https://desarrolloweb.com/articulos/2477.php>

**Recuperado por Manyari Roblero Gil el 27 de abril de 2020:** <https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/>

**Recuperado por Jhonatan Gómez García el 27 de abril de 2020:** PIATTINI, Mario. Diseño de Base de Datos Relacionales. Edit. Alfaomega.