# Requisitos

👨‍🎓 Requisitos 👩‍🎓 :

requisitos

Los requisitos necesarios para seguir este curso serían tener instalados los siguientes softwares:

- 🔗 Beckhoff TwinCAT 3 XAE ó el IDE de 🔗 Codesys.
- Tener cuenta de usuario creada en 🔗 GitHub.
- saber lo mínimo de Git o apoyarse en herramientas visuales como pueden ser:
  - 🔗 GitHub Desktop.
  - 🔗 sourcetree
  - 🔗 tortoiseGit, etc...
  - Sería bueno tener algo de conocimientos previos de teoria de OOP, aunque sean en otros lenguajes de programación ya que seran extrapolables para el enfoque de este curso de OOP IEC61131-3 para PLCs.

## Pasos para empezar:

- Clonar el repositorio de 🔗GitHub:

  $ git clone https://github.com/runtimevic/OOP-IEC61131-3--Curso-Youtube.git

  ó utilizar por ejemplo GitHub Desktop para Clonar el repositorio de GitHub...

- Nos encontraremos las siguientes carpetas:
  - 🔗 TC3_OOP: Dentro de esta carpeta se encuentra el proyecto de TwinCAT3, con todo lo que se va explicando en los videos de youtube...
  - 🔗 Ficheros_PLCOpen_XML: Dentro de esta carpeta nos iremos encontrando los ficheros exportados en formato PLCOpen XML para que puedan ser importados en TwinCAT3 ó en Codesys de todo lo explicado en Youtube, ya que al ser el formato standarizado de PLCOpen se puede exportar/importar en todas las marcas de PLCs que sigan el estandard PLCOpen..., pero es recomendable intentar realizar lo explicado desde cero para ir practicando y asumir los conceptos explicados...
  - tambien esta alojada la creación de esta pagina web SSG, (Generador de Sitios Estáticos) la cual se ira modificando conforme avancemos en este Curso de OOP IEC-61131-3 PLC...

# Introdução

📖 Curso Programación Orientada a Objetos Youtube -- OOP :

OOP_Titulo by Runtimevic -- Víctor Durán Muñoz.

---

## ¿ Qué es OOP?

- Es un paradigma que hace uso de los objetos para la construcción de los software.

  . ¿ Qué es un paradigma?

  - Tiene diferentes interpretaciones, puede ser un **modelo**, **ejemplo** o **patrón.**
  - Es una **forma** o un **estilo** de programar.
  - se busca plasmar la realidad hacia el código.

## ¿Cómo pensar en Objetos?

- Enfocarse en **algo de la realidad.**
- Detalla sus **atributos, (propiedades)**
- Detalla sus **comportamientos (metodos)**

```
📱 Ejemplo: (Telefono móvil-smartphone)

. ¿Qué atributos (Propiedades) reconocemos?
    - color.
    - marca.
. ¿Qué se puede hacer? (Metodos)
    - Realizar llamadas.
    - Navegar por internet.
```

```
🚗 Ejemplo: (Coche)

. ¿Qué atributos (Propiedades) reconocemos?
    - color.
    - marca.
. ¿Qué se puede hacer? (Metodos)
    - conducir.
    - frenar.
    - acelerar.
```

---

# Tipos de paradigmas

## Tipos de paradigmas:

- Imperativa -- (**Instrucciones a seguir** para dar solución a un problema).
- Declarativa -- (Se **enfoca en el problema** a solucionar).
- Estructurada -- (La solución a un problema sigue **una secuencia de inico a fin**).
- Funcional -- (Divide el problema en diversas soluciones que serán ejecutadas por las **funciones declaradas**). La programación procedimental o programación por procedimientos es un paradigma de la programación. Muchas veces es aplicable tanto en lenguajes de programación de bajo nivel como en lenguajes de alto nivel. En el caso de que esta técnica se aplique en lenguajes de alto nivel, recibirá el nombre de programación funcional.
    - se llaman rutinas separadas desde el programa principal
    - datos en su mayoría globales -> sin protección.
    - los procedimientos por lo general no pueden ser independientes -> mala reutilización del código.

programacion_procedimental

- Orientada a objetos -- Construye soluciones **basadas en objetos**.

```
wikipedia:
La programación orientada a objetos es un paradigma de programación
basado en el concepto de "objetos", que pueden contener datos y
código.
Los datos están en forma de campos y el código está en forma de
procedimientos.
```

ventajasprogramacionoop

## Ventajas de la Programación OOP:

- rutinas y datos se combinan en un objeto -> Encapsulación.
- métodos/Propiedades -> interfaces definidas para llamadas y acceso a datos.

---

# Tipos de Dados

## Declaracion de una Variable:

La declaración de variables en CODESYS ó TwinCAT incluirá:

- Un nombre de variable
- Dos puntos
- Un tipo de dato
- Un valor inicial opcional
- Un punto y coma
- Un comentario opcional

```
( <pragma> )*
<scope> ( <type qualifier> )?
      <identifier> (AT <address> )? : <data type> ( := <initial
value> )? ;
END_VAR
```

- 🔗 infosys.beckhoff.com, Declaring variables

```
VAR
    nVar1   : INT := 12;                    // initialization value 12
    nVar2   : INT := 13 + 8;                // initialization value
defined by an expression of constants
    nVar3   : INT := nVar2 + F_Fun(4);      //initialization value defined
by an expression that contains a function call; notice the order!
    pSample : POINTER TO INT := ADR(nVar1); //not described in the
standard IEC61131-3: initialization value defined by an adress function;
Notice: the pointer will not be initialized during an Online Change
END_VAR
```

---

## 🔗 Tipos de Datos:

### Las ventajas de las estructuras de datos.

- La principal aportación de las estructuras de datos y los tipos de datos creados por el usuario es la claridad y el orden del código resultante.

---

# Tipos de variáveis e variáveis especiais

## Variable types and special variables:

The variable type defines how and where you can use the variable. The variable type is defined during the variable declaration.

## Further Information:

- Local Variables - VAR
  - Las variables locales se declaran en la parte de declaración de los objetos de programación entre las palabras clave VAR y END_VAR.
  - Puede extender las variables locales con una palabra clave de atributo.
  - Puede acceder a variables locales para leer desde fuera de los objetos de programación a través de la ruta de instancia. El acceso para escribir desde fuera del objeto de programación no es posible; Esto será mostrado por el compilador como un error.
  - Para mantener la encapsulación de datos prevista, se recomienda encarecidamente no acceder a las variables locales de un POU desde fuera del POU, ni en modo de lectura ni en modo de escritura. (Otros compiladores de lenguaje de alto nivel también generan operaciones de acceso de lectura a variables locales como errores). Además, con los bloques de funciones de biblioteca no se puede garantizar que las variables locales de un bloque de funciones permanezcan sin cambios durante las actualizaciones posteriores. Esto significa que es posible que el proyecto de aplicación ya no se pueda compilar correctamente después de la actualización de la biblioteca.
  - También observe aquí la regla SA0102 del Análisis Estático, que determina el acceso a las variables locales para la lectura desde el exterior.
- Input Variables - VAR_INPUT
  - Las variables de entrada son variables de entrada para un bloque de funciones.
  - VAR_INPUT variables se declaran en la parte de declaración de los objetos de programación entre las palabras clave VAR_INPUT y END_VAR.
  - Puede ampliar las variables de entrada con una palabra clave de atributo.
- Output Variables - VAR_OUTPUT
  - Las variables de salida son variables de salida de un bloque de funciones.
  - VAR_OUTPUT variables se declaran en la parte de declaración de los objetos de programación entre las palabras clave VAR_OUTPUT y END_VAR. TwinCAT devuelve

# Modificadores de acesso

## Modificadores de Acceso:

- PUBLIC:
  - Son accesibles luego de instanciar la clase.
  - Corresponde a la especificación de modificador sin restricción de acceso.
- PRIVATE:
  - Son accesibles solo dentro de la clase.
  - El acceso está restringido al bloque de funciones Heredado y en el programa MAIN, respectivamente.
- PROTECTED:
  - Son accesibles dentro de la clase.
  - Son accesibles a través de la herencia.
  - El acceso está restringido, no se puede acceder desde el programa principal, desde el MAIN.
- INTERNAL:
  - El acceso está limitado al espacio de nombres (la biblioteca).
- FINAL:
  - No se permite sobrescribir, en un derivado del bloque de funciones.
  - Esto significa que no se puede sobrescribir/extender en una subclase posiblemente existente.

# Tabela de Modificadores de acesso

| Modificadores de acceso | FUNCTION_BLOCK - FB | METODO | PROPIEDAD |
|---|---|---|---|
| **PUBLIC** | Si | Si | Si |
| **INTERNAL** | Si | Si | Si |
| **FINAL** | Si | Si | Si |
| **ABSTRACT** | Si | Si | Si |
| **PRIVATE** | No | Si | Si |
| **PROTECTED** | No | Si | Si |

# Texto Estruturado Estendido

## ExST - Texto Estructurado Extendido:

---

- 🔗 Structured Text and Extended Structured Text (ExST), infosys.beckhoff.com

# 4 Pilares

Principios OOP: (4 pilares)

- Abstracción -- La forma de **plasmar algo hacia el código** para enfocarse en su uso. No enfocarnos tanto en que hay por detras del codigo si no en el uso de este.
- Encapsulamiento -- No toda la información de nuestro objeto es **relevante y/o accesible** para el usuario.
- Herencia -- Es la cualidad de **heredar caracteristicas** de otra clase. (**EXTENDS**)
- Polimorfismo -- Las **múltiples formas** que puede obtener un objeto si comparte la misma **clase o interfaz**. (**IMPLEMENTS**)

OOP_Principios

---

## Links de Principios OOP:

- 🔗 github.com/Aliazzzz/OOP-Concept-Examples-in-CODESYS-V3

# Abstração

## ABSTRACCION:

La Abstracción es el proceso de ocultar información importante, mostrando solo lo información más esencial. Reduce la complejidad del código y aísla el impacto de los cambios. La abstracción se puede entender a partir de un ejemplo de la vida real: encender un televisor solo debe requieren hacer clic en un botón, ya que las personas no necesitan saber el proceso por el que pasa. Aunque ese proceso puede ser complejo e importante, no es necesario que el usuario sepa cómo se implementa. La información importante que no se requiere está oculta para el usuario, reduciendo la complejidad del código, mejorando la ocultación de datos y la reutilización, haciendo así que los Bloques de Funciones sean más fáciles de implementar y modificar.

La palabra clave ABSTRACT está disponible para bloques de funciones, métodos y propiedades. Permite la implementación de un proyecto PLC con niveles de abstracción.

La abstracción es un concepto clave de la programación orientada a objetos. Los diferentes niveles de abstracción contienen aspectos de implementación generales o específicos.

## Aplicación de la abstracción:

Es útil implementar funciones básicas o puntos en común de diferentes clases en una clase básica abstracta. Se implementan aspectos específicos en subclases no abstractas. El principio es similar al uso de una interfaz. Las interfaces corresponden a clases puramente abstractas que contienen sólo métodos y propiedades abstractas. Una clase abstracta también puede contener métodos y propiedades no abstractos.

Reglas para el uso de la palabra clave ABSTRACT:

- No se pueden instanciar bloques de funciones abstractas.
- Los bloques de funciones abstractas pueden contener métodos y propiedades abstractos y no abstractos.
- Los métodos abstractos o las propiedades no contienen ninguna implementación (sólo la declaración).
- Si un bloque de función contiene un método o propiedad abstracta, debe ser abstracto.
- Los bloques de funciones abstractas deben extenderse para poder implementar los métodos o propiedades abstractos.
- Por lo tanto: un FB derivado debe implementar los métodos/propiedades de su FB básico o también debe definirse como abstracto.

# Encapsulamento

## Encapsulamiento:

La encapsulación se utiliza para agrupar datos con los métodos que operan en ellos y para ocultar datos en su interior. una clase, evitando que personas no autorizadas accedan directamente a ella. Reduce la complejidad del código y aumenta la reutilización. La separación del código permite la creación de rutinas que pueden ser reutilizadas en lugar de copiar y pegar código, reduciendo la complejidad del programa principal.

---

## Links Encapsulacion:

- 🔗 www.plccoder.com,encapsulation

# Herança

## Herencia:

La herencia permite al usuario crear clases basadas en otras clases. Las clases heredadas pueden utilizar las funcionalidades de la clase base, así como algunas funcionalidades adicionales que el usuario puede definir. Elimina el código redundante, evita copiar y pegar y facilita la expansión. Esto es muy útil porque permite ampliar o modificar (anular) las clases sin cambiar la implementación del código de la clase base. ¿Qué tienen en común un teléfono fijo antiguo y un smartphone? Ambos pueden ser clasificados como teléfonos. ¿Deberían clasificarse como objetos? No, ya que también definen las propiedades y comportamientos de un grupo de objetos. Un teléfono inteligente funciona como un teléfono normal, pero también es capaz de tomar fotografías, navegar por Internet y hacer muchas otras cosas. Entonces, teléfono fijo antiguo y el teléfono inteligente son clases secundarias que amplían la clase de teléfono principal.

---

- Superclase: La clase cuyas características se heredan se conoce como superclase (ó una clase base ó una clase principal ó clase padre).
- Subclase: La clase que hereda la otra clase se conoce como subclase (ó una clase derivada, clase extendida ó clase hija).

---

## Links Herencia:

- 🔗 stefanhenneken.net,iec-61131-3-methods-properties-and-inheritance

# Polimorfismo

## Polimorfismo:

El concepto de polimorfismo se deriva de la combinación de dos palabras: Poly (Muchos) y Morfismo (Forma). Refactoriza casos de cambio/declaraciones de casos feos y complejos. El polimorfismo permite que un objeto cambie su apariencia y desempeño dependiendo de la situación práctica para poder realizar una determinada tarea. Puede ser estático o dinámico:

- El polimorfismo estático ocurre cuando el compilador define el tipo de objeto;
- El polimorfismo dinámico se produce cuando el tipo se determina durante el tiempo de ejecución, lo que hace posible para que una misma variable acceda a diferentes objetos mientras el programa se está ejecutando. Un buen ejemplo para explicar el polimorfismo es una navaja suiza. Una navaja suiza es una herramienta única que incluye un montón de recursos que se pueden utilizar para resolver problemas diferentes. Al seleccionar la herramienta adecuada, se puede utilizar una navaja suiza para realizar un determinado conjunto de tareas valiosas. De la manera dual, un bloque sumador simple que se adapta para hacer frente a, por ejemplo, los tipos de datos int, float, string y time es un ejemplo de un polimórfico recurso de programación.

---

## ¿Como conseguir el Polimorfismo?

El polimorfismo se puede obtener gracias a las Interfaces y/o las Clases Abstractas.

- Interface: (INTERFACE)
  - Son un **contrato que obliga** a una clase a **implementar** las **propiedades** y/o **métodos** definidos.
  - Son una plantilla (sin lógica).
- Clases Abstractas: (ABSTRACT)
  - Son Clases que no se pueden instanciar, solo pueden ser implementadas a través de la herencia.
- Diferencias:

| **Clases abstractas** | **Interfaces** |
| --- | --- |
| 1.- Limitadas a una sola implementación. | 1. No tiene limitación de implementación. |

# UML

## UML (Unified Modeling Language) o “Lenguaje Unificado de Modelado”:

### Descripción general:

UML (Lenguaje de modelado unificado) es un lenguaje gráfico para la especificación, diseño y documentación de software orientado a objetos. Proporciona una base universalmente comprensible para la discusión entre la programación y otros departamentos dentro del desarrollo del sistema.

El lenguaje de modelado unificado en sí mismo define 14 tipos de diagramas diferentes de dos categorías principales:

### Diagramas de Estructura:

Los diagramas de estructura representan esquemáticamente la arquitectura del software y se utilizan principalmente para modelado y análisis (por ejemplo, diseño de proyectos, especificación de requisitos del sistema y documentación).

En Codesys y en TwinCAT tendremos el diagrama:

- Diagrama de clase UML (tipo: diagrama de estructura)

### Diagramas de Comportamiento:

Los diagramas de comportamiento son modelos ejecutables con sintaxis y semántica únicas a partir de los cuales se puede generar directamente el código de la aplicación (arquitectura basada en modelos).

En Codesys y en TwinCAT tendremos el diagrama:

- UML Statechart (tipo: diagrama de comportamiento)

---

### Links UML:

- 🔗 Tutorial UML en español
- 🔗 www.plccoder.com, twincat-uml-class-diagram
- 🔗 content.helpme-codesys.com, uml_general
- 🔗 content.helpme-codesys.com, Class Diagram Elements

# Class UML

## Diagrama de Clases en UML:

La jerarquía de herencia se puede representar en forma de diagrama. El lenguaje de modelado unificado (UML) es el estándar establecido en esta área. UML define varios tipos de diagramas que describen tanto la estructura como el comportamiento del software.

Una buena herramienta para describir la jerarquía de herencia de bloques de funciones es el diagrama de clases.

Los diagramas UML se pueden crear directamente en TwinCAT 3. Los cambios en el diagrama UML tienen un efecto directo en las POU. Por lo tanto, los bloques de funciones se pueden modificar y modificar a través del diagrama UML.

Cada caja representa un bloque de función y siempre se divide en tres secciones horizontales. La sección superior muestra el nombre del bloque de funciones, la sección central enumera sus propiedades y la sección inferior enumera todos sus métodos. En este ejemplo, las flechas muestran la dirección de la herencia y siempre apuntan hacia el bloque de funciones principal.

---

Los Modificadores de acceso de los metodos y las propiedades se veran segun la simbologia: (Disponible a partir de la versión de TwinCAT 3.1.4026)

UML_ClassDiagram Access Modifier

## Links UML listado de referencias:

- 🔗 stefanhenneken.net, UML Class
- 🔗 www.lucidchart.com/tutorial-de-diagrama-de-clases-uml
- 🔗 www.edrawsoft.com/uml-class-diagram-explained
- 🔗 blog.visual-paradigm.com/what-are-the-six-types-of-relationships-in-uml-class-diagrams
- 🔗 Ingeniería del Software: Fundamentos de UML usando Papyrus
- 🔗 plantuml.com/class-diagram
- 🔗 www.planttext.com
- 🔗 UML Infosys Beckhoff
- 🔗 Tutorial - Diagrama de Clases UML

# Relações

.Relaciones:

Vamos a ver 2 tipos de relaciones:

- Asociación.
  - De uno a uno: Una clase mantiene una **asociación de a uno** con otra clase.
  - De uno a muchos: Una clase mantiene una asociación con otra clase **a través de una colección**.
  - De muchos a muchos: La **asociación se da en ambos lados** a través de una colección.
- Colaboración.
  - La colaboración se da **a través de una referencia de una clase** con el fin de **lograr un cometido**.

# StateChart UML

## UML State Chart:

---

## Links UML State Chart:

- 🔗 content.helpme-codesys.com, UML State Chart

# Tipos de Design para programação de PLC

## Tipos de Diseño para programacion de PLC:

Ingenieria de desarrollo para la programación OOP - Diseño por Componente, Unidad, Dispositivo, Objeto... - Los objetos son las unidades básicas de la programación orientada a objetos. - Un componente proporciona servicios, mientras que un objeto proporciona operaciones y métodos. Un componente puede ser entendido por todos, mientras que un objeto solo puede ser entendido por los desarrolladores. - Las unidades son los grupos de código más pequeños que se pueden mantener y ejecutar de forma independiente - Diseño por Pruebas Unitarias. - Diseño en UML.

Units: (Ejemplo de Unidades): - I_InputDigital(p_On, p_Off) - I_OutputDigital(M_ON, M_OFF) - I_InputAnalog - I_OutputAnalog - I_Run:(M_Start, M_Stop)

-FBTimer -FCAnalogSensor -FBGenericUnit

!!! puntos que se pueden incluir en el curso!!!: - Objects composition (Composicion de Objetos)

- Basic of Structured Text programming Language
- UDT (estructuras)
- Modular Design
- Polymorphism
- Advanced State Pattern
- Wrappers and Features
- Layered Design
- Final Project covering a real-world problem to be solved using OOP
- Texto estructurado (ST), Texto estructurado extendido (ExST)

# Padrões de Design

## PATRONES DE DISEÑO:

Designpatterns

Los patrones de diseño son soluciones generales y reutilizables para problemas comunes que se encuentran en la programación de software. En la programación orientada a objetos, existen muchos patrones de diseño que se pueden aplicar para mejorar la modularidad, la flexibilidad y el mantenimiento del código. Algunos ejemplos de patrones de diseño que se pueden aplicar en la programación de PLCs incluyen el patrón Singleton, el patrón Factory Method, el patrón Observer y el patrón Strategy. Por ejemplo, el patrón Singleton se utiliza para garantizar que solo exista una instancia de una clase determinada en todo el programa. Esto puede ser útil en la programación de PLCs cuando se quiere asegurar que solo hay una instancia activa del objeto que controla un determinado proceso o dispositivo. El patrón Factory Method se utiliza para crear instancias de objetos sin especificar explícitamente la clase concreta a instanciar. Esto puede ser útil en la programación de PLCs cuando se quiere crear objetos según las necesidades específicas del programa. El patrón Observer se utiliza para establecer una relación uno a muchos entre objetos, de manera que cuando un objeto cambia su estado, todos los objetos relacionados son notificados automáticamente. Este patrón puede ser muy útil en la programación de PLCs para establecer relaciones entre diferentes componentes del sistema, como sensores y actuadores. El patrón Strategy se utiliza para definir un conjunto de algoritmos intercambiables, y luego encapsular cada uno como un objeto. Este patrón puede ser útil en la programación de PLCs cuando se desea cambiar dinámicamente el comportamiento del sistema según las condiciones del entorno. En resumen, los patrones de diseño son una herramienta muy útil para mejorar la calidad del código en la programación de PLCs y se pueden aplicar con éxito en la programación orientada a objetos para PLCs.

```
1  “Los patrones de diseño son
2  descripciones de objetos y clases
3  conectadas que se personalizan para
4  resolver un problema de diseño
5  general en un contexto particular”.
6  - Gang of Four
```

Patrones_de_Diseño_Creacional

Patrones_de_Diseño_Estructural

Patrones_de_Diseño_de_Comportamiento

Design_patterns

# Padrão Método de fábrica

---

Links de Patrones de Diseño:

- 🔗 Factory Method Design Pattern

# Padrão Fábrica Abstrata

---

Links de Patrones de Diseño:

- 🔗 iec-61131-6-abstract-factory-english,stefanhenneken.net
- 🔗 Abstract Factory Design Pattern
- 🔗 refactoring.guru,abstract-factory

# Padrão Decorador

## Links de Patrones de Diseño:

- 🔗 Decorator Design Pattern

# Padrão de Estratégia

---

Links de Patrones de Diseño:

- 🔗 TwinCAT with Head First Design Patterns Ch.1 - Intro/Strategy Pattern
- 🔗 PATRONES de DISEÑO en PROGRAMACIÓN FUNCIONAL

---

# Padrão Observador

---

Links de Patrones de Diseño:

- 🔗 Observer Design Pattern

# Padrão Visitante

---

Links de Patrones de Diseño:

- 🔗Design Patterns - The Visitor Pattern

# Livrarias

## Librerias:

Cuando desarrollas un proyecto, ¿qué haces cuando quieres reutilizar el mismo programa para otro proyecto? Probablemente el más común es copiar y pegar. Esto está bien para proyectos pequeños, pero a medida que crece la aplicación, las bibliotecas nos permiten administrar las funciones y los bloques de funciones que hemos creado.

Mediante el uso de bibliotecas, podemos administrar el software que hemos creado en múltiples proyectos. En primer lugar, es un hecho que diferentes dispositivos tendrán diferentes funciones, pero aun así, siempre habrá partes comunes. En el mundo del desarrollo de software, ese concepto de gestión de bibliotecas es bastante común.

## ¿Cuáles son las ventajas de usar la biblioteca?

- El software es modular, por ejemplo, si tengo software para cilindros, puedo usar la biblioteca de cilindros, y si tengo software para registro, puedo usar la biblioteca de registro.
- Cada biblioteca se prueba de forma independiente.

---

## Links Librerias:

- 🔗 soup01.com,beckhofftwincat3-library-management
- 🔗 PLC programming using TwinCAT 3 - Libraries (Part 11/18)
- 🔗 help.codesys.com,_cds_obj_library_manager/
- 🔗 help.codesys.com,_cds_library_development_information/
- 🔗 help.codesys.com,_tm_test_action_libraries_addlibrary
- 🔗 CODESYS Webinar Library Management Basics
- 🔗 CoDeSys - How to add libraries and more with Machine Control Studio.

# Links OOP

Links de OOP:

Mención a la Fuentes Links empleadas para la realización de esta Documentación:

# TDD - Desenvolvimento Orientado a Testes

## Desarrollo Guiado por Pruebas:

La clave del TDD o Test Driven Development es que en este proceso se escriben las pruebas antes de escribir el código. Este sistema consigue no solo mejorar la calidad del software final, sino que, además, ayuda a reducir los costes de mantenimiento.

La premisa detrás del desarrollo dirigido por pruebas, según Kent Beck, es que todo el código debe ser probado y refactorizado continuamente. Kent Beck lo expresa de esta manera: **Sencillamente, el desarrollo dirigido por pruebas tiene como objetivo eliminar el miedo en el desarrollo de aplicaciones**.

- Está creando documentación, especificaciones vivas y nunca obsoletas (es decir, documentación).
- Está (re)diseñando su código para hacerlo y mantenerlo fácilmente comprobable. Y eso lo hace limpio, sin complicaciones y fácil de entender y cambiar.
- Está creando una red de seguridad para hacer cambios con confianza.
- Notificación temprana de errores.
- Diagnóstico sencillo de los errores, ya que las pruebas identifican lo que ha fallado.

## El Ciclo y las Etapas del TDD:

El Desarrollo Dirigido por Pruebas significa pasar por tres fases. Una y otra vez.

- Fase roja: escribir una prueba.
- Fase verde: hacer que la prueba pase escribiendo el código que vigila.
- Fase azul: refactorizar.

## Niveles de Testing:

Levels of Testing

The_Pyramid_Of_Test

---

- El Desarrollo Guiado por Pruebas debe de empezarse a implementar lo mas temprano posible en el desarrollo del Software.

# Testes de Unidade