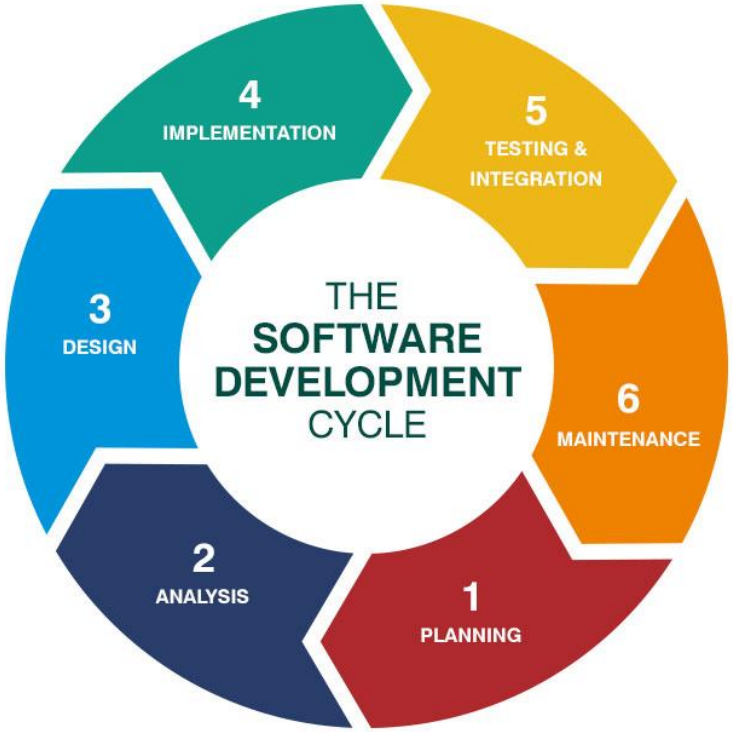
# Introducción: Proceso Desarrollo de Software

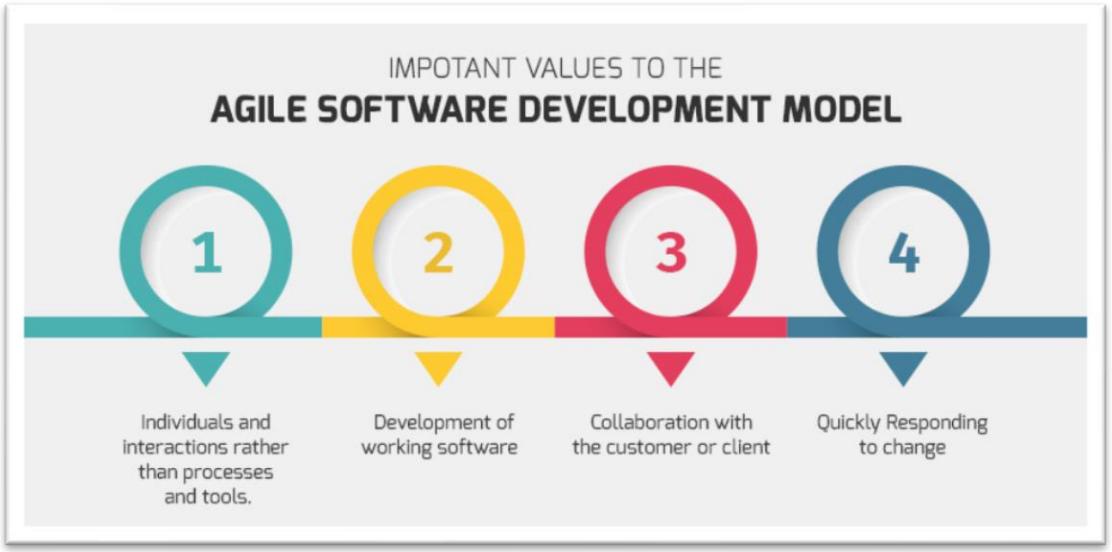


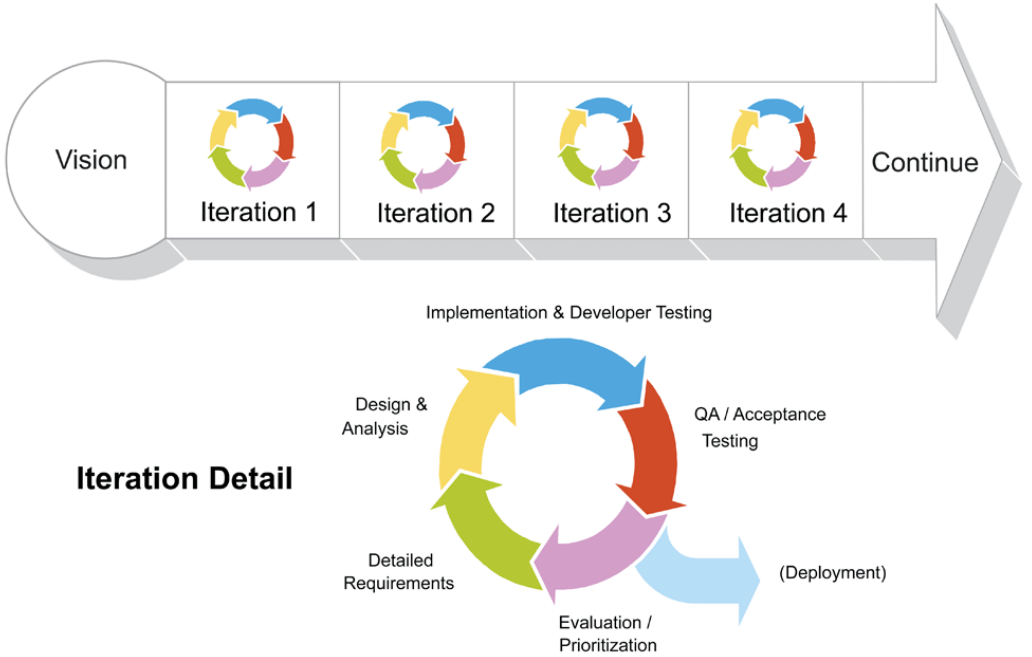

- Definición de este concepto
- Principales metodologías de Desarrollo
- Introducción a la metodología RUP
- ¿Qué es SCRUM?
- Principios de desarrollo de software
- Patrones de diseño de software

**Tecnología de objetos**
**Profesor:** Samuel Luciano Lassis

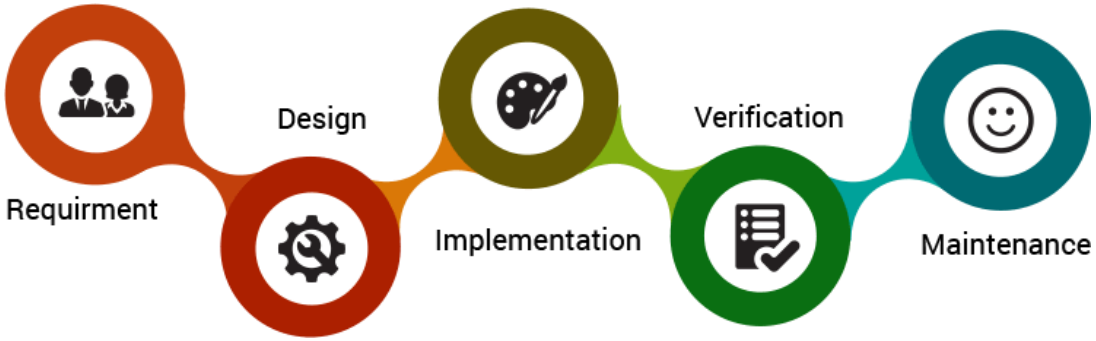
"El **proceso de desarrollo de software** describe la forma sobre la cual podemos construir, desarrollar y posiblemente mantener un sistema de computación."
(Larman, 2004)

# Procesos de desarrollo de software



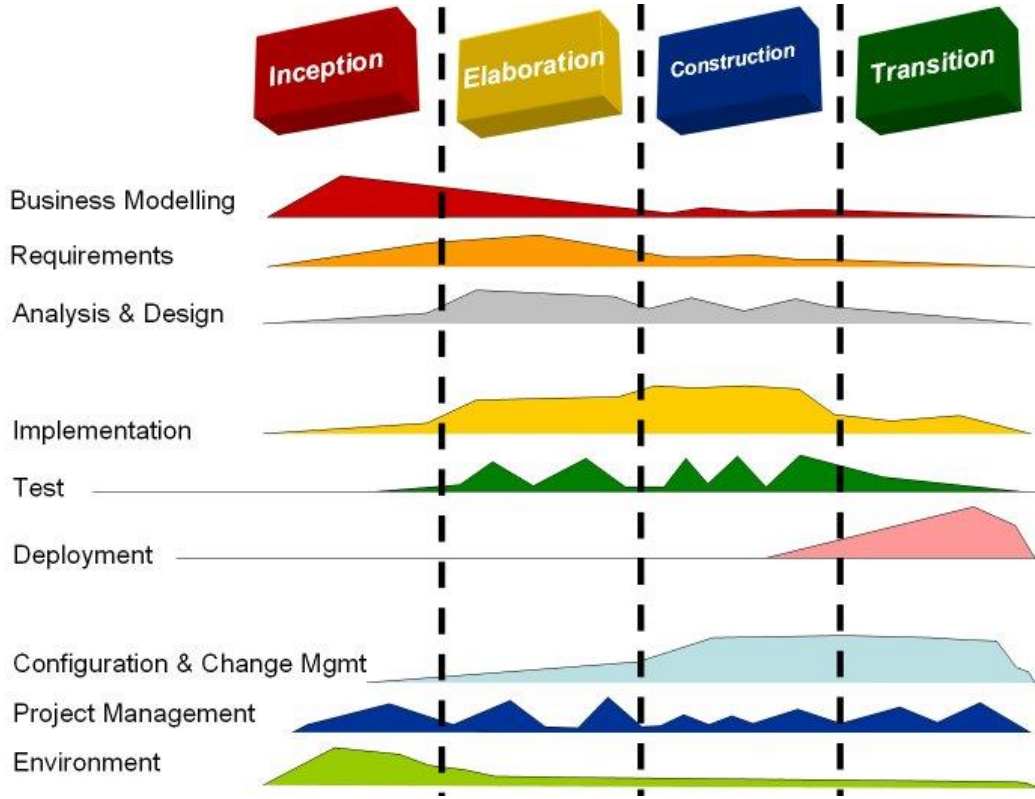

## Incremental | Iterativo




## Cascada (Waterfall)

# Rational Unified Process (RUP)

Es una metodología de desarrollo de software iterativa que utiliza la retroalimentación de las iteraciones anteriores para mejorar el proceso. RUP busca reducir el riesgo del proyecto colocando los requerimientos de mayor riesgos delante de aquellos requerimientos que presenten menor riesgo para el cliente y ayuda a construir prototipos y diseños de flujo de interfaces de usuario de manera que el cliente conozca desde el inicio y pueda identificar de manera mas precisa los riesgos.



# Rational Unified Process (RUP): Etapas

## Inception

- Comunicación con el cliente para captura de requerimientos
- Propósito y objetivo del proyecto
- Análisis de factibilidad técnica, económica, social, Ambiental
- Decisiones de comprar o hacer
- Estimación de costos y tiempo de ejecución
- Documento de especificación de requerimientos (SRS)
- Diagrama de casos de uso
- Documento de especificaciones no funcionales
- Glosario de términos

## Elaboration

- Análisis de requerimientos y diseño de arquitecura
- Modelos creados desde la perspectiva de la vista lógica: modelos de casos de uso y modelos de dominio
- Los arquitectos definen los patrones arquitectonicos que serán utilizados en el Desarrollo
- Diagramas de casos de uso
- Casos de uso detallados
- Diagrama de clases
- Diagrama de secuencias
- Diagrama de estados

## Construction

- El desarrollo y pruebas se inician
- RUP, en esta etapa, alienta la reusabilidad en el desarrollo
- Componentes son desarrollados basado en OOP y sus buenas practices
- Pruebas unitarias de componentes desarrollados
- Desarrollo de pruebas de integración
- Código Fuente
- Reporte de resultados de pruebas

## Transition

- Errores menores al 0.01%
- Entrega del software y sus fucnionalidades
- Pruebas de aceptación de usuarios: Alpha y Beta
- Manuales de usuario
- Ejecutables
- Repositorio de casos de prueba

"**Agile development** methods usually apply timeboxed iterative and evolutionary development, employ adaptive planning, promote incremental delivery, and include other values and practices that encourage agility, rapid and flexible response to changes." (Larman, 2004)

# Manifiesto Agil

**Individuos e interacciones**
sobre procesos y herramientas

**Software funcionando**
sobre documentación extensiva

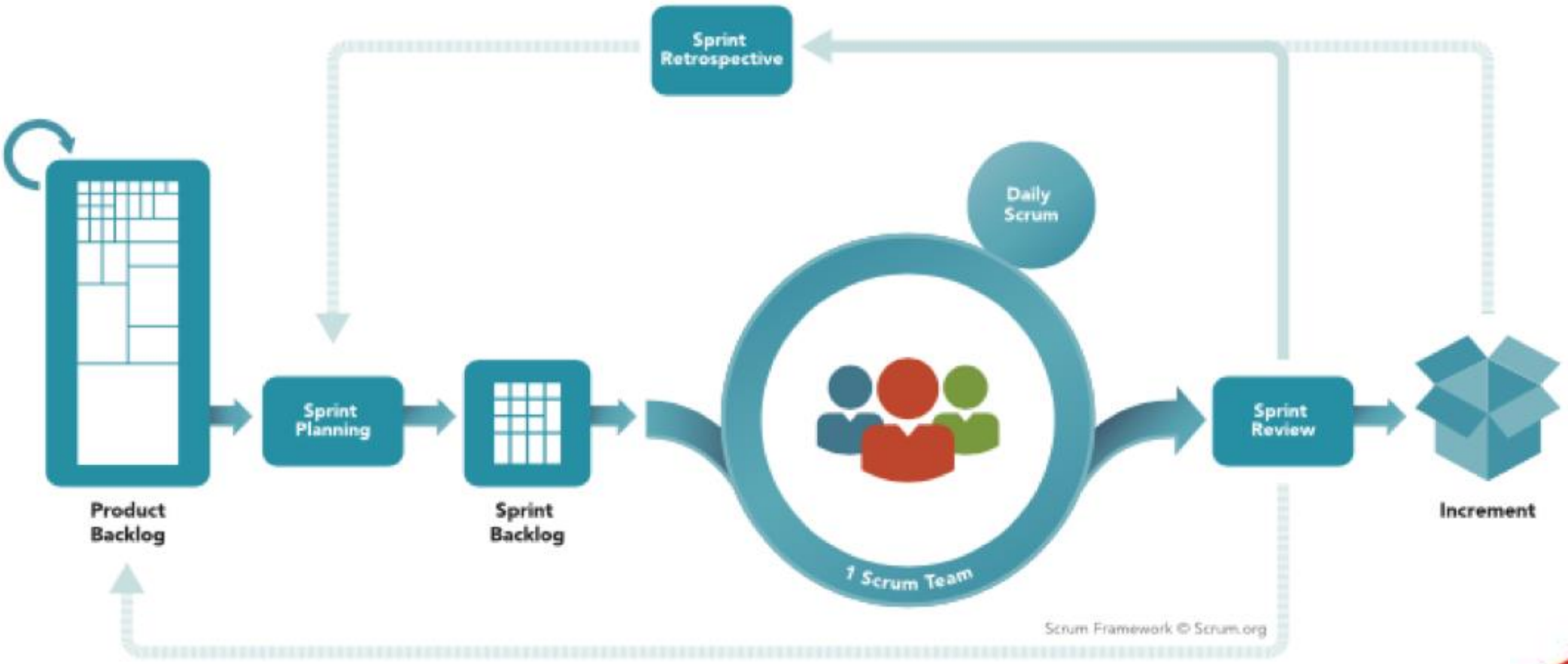
**Colaboración con el cliente**
sobre negociación contractual

**Respuesta ante el cambio**
sobre seguir un plan

# Manifiesto Ágil: Principios

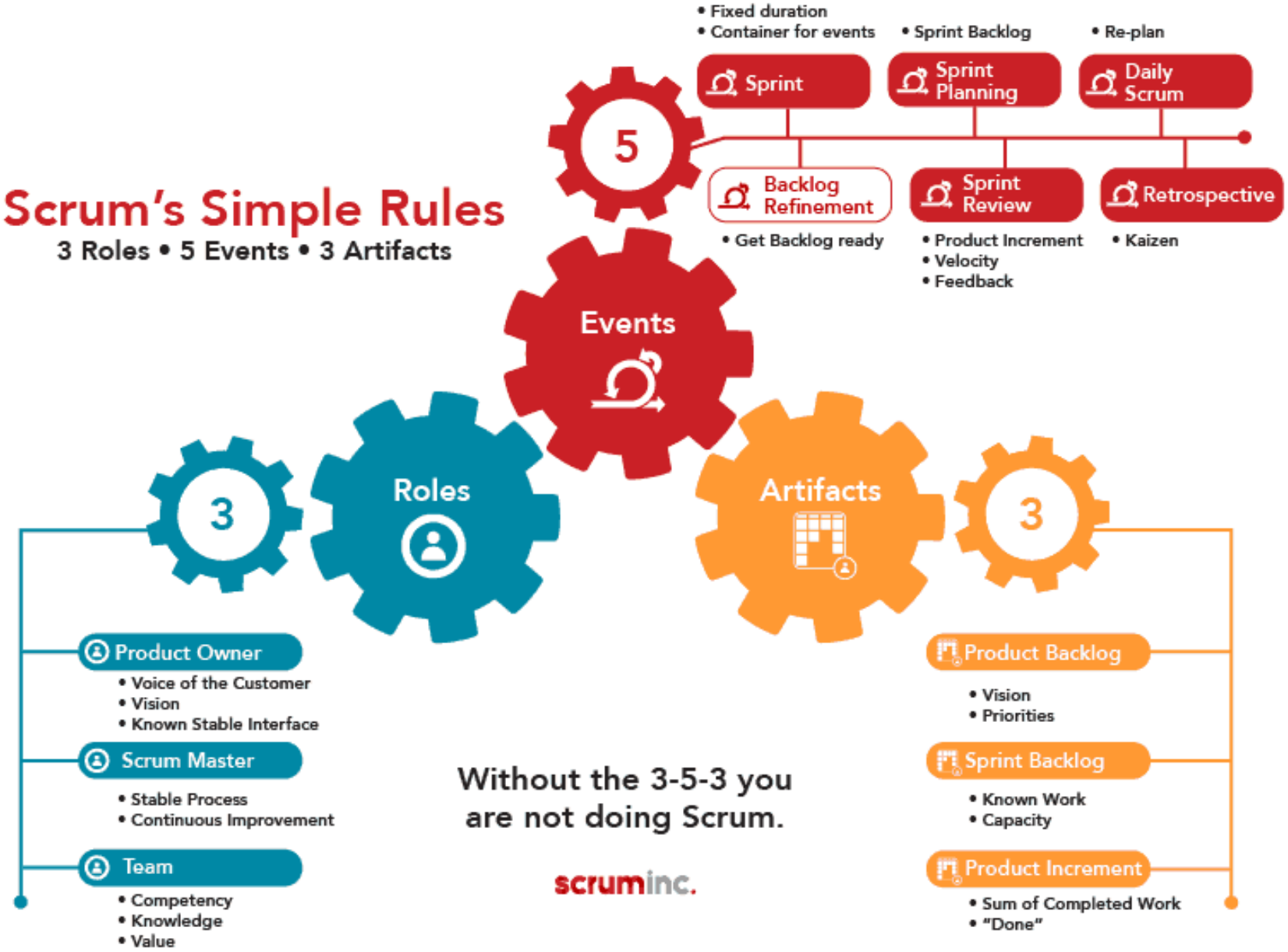
- Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Aceptamos que los requisitos cambien, incluso en etapas  tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de  tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

# SCRUM FRAMEWORK



# Scrum's Simple Rules

3 Roles • 5 Events • 3 Artifacts




Without the 3-5-3 you are not doing Scrum.

scruminc.

“Writing a code which satisfies present requirements, also which can satisfy future requirement easily, should be the goal of every developer.” - Raj Suvariya

# Principios de desarrollo de software: S.O.L.I.D

**Single Responsibility Principle (SRP)**

“class should be having one and only one responsibility”

**Open/Close Principle**

“classes should be open for extension but closed for modification”

**Liskov’s Substitution Principle**

"parent classes should be easily substituted with their child classes without blowing up the application"

**Interface Segregation Principle**

"many client specific interfaces are better than one general interface"

**Dependency Inversion Principle**

“classes should depend on abstraction but not on concretion”

"**Design pattern** is a general reusable solution or template to a commonly occurring problem in software design. The patterns typically show relationships and interactions between classes or objects. The idea is to speed up the development process by providing tested, proven development paradigm." (GeeksForGeeks, s.f.)

GeeksForGeeks (s.f.). Design Patterns | Set 1 (Introduction).
Recuperado de https://www.geeksforgeeks.org/design-patterns-set-1-introduction/

# Patrones de diseño: tipos

**De creación:** Son aquellos patrones de diseño que nos permiten la creación/instanciación de nuevos objetos dentro de OOP

- Factory Method
- Singleton
- Abstract Method
- …

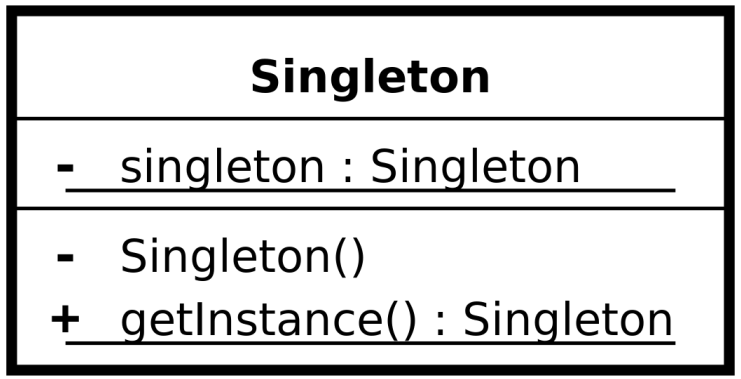
**De estructura:** Permiten la organización de clases y objetos para formar estructuras largas y complejas que modelan sistemas

- Adapter
- Composite
- Facade
- …

**De comportamiento:** Modelan comportamientos comúnes de comunicación entre objetos

- Iterator
- Strategy
- Observer
- …

# Singleton Pattern




## Diagrama UML

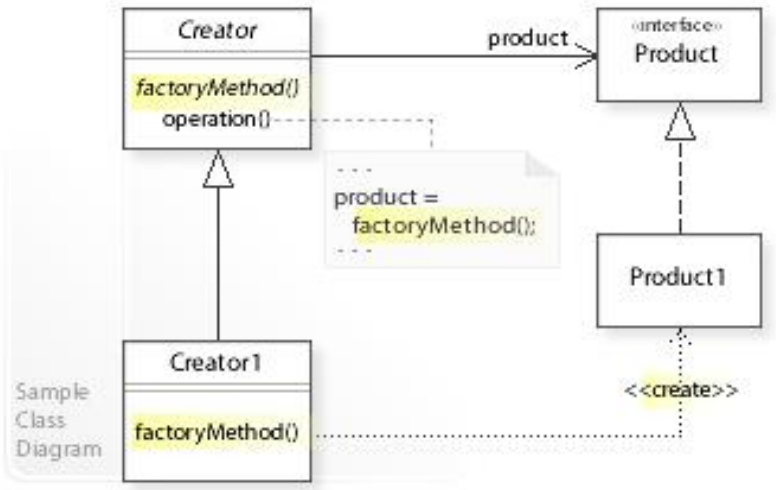
## Implementación

```
// Classical Java implementation of singleton
// design pattern
class Singleton
{
    private static Singleton obj;

    // private constructor to force use of
    // getInstance() to create Singleton object
    private Singleton() {}

    public static Singleton getInstance()
    {
        if (obj==null)
            obj = new Singleton();
        return obj;
    }
}
```

# Factory Method Pattern




## Diagrama UML

```
//Empty vocabulary of actual object
public interface IPerson
{
    string GetName();
}

public class Villager : IPerson
{
    public string GetName()
    {
        return "Village Person";
    }
}

public class CityPerson : IPerson
{
    public string GetName()
    {
        return "City Person";
    }
}

public enum PersonType
{
    Rural,
    Urban
}

/// <summary>
/// Implementation of Factory - Used to create objects
/// </summary>
public class Factory
{
    public IPerson GetPerson(PersonType type)
    {
        switch (type)
        {
            case PersonType.Rural:
                return new Villager();
            case PersonType.Urban:
                return new CityPerson();
            default:
                throw new NotSupportedException();
        }
    }
}
```

## Implementación

# Strategy Pattern

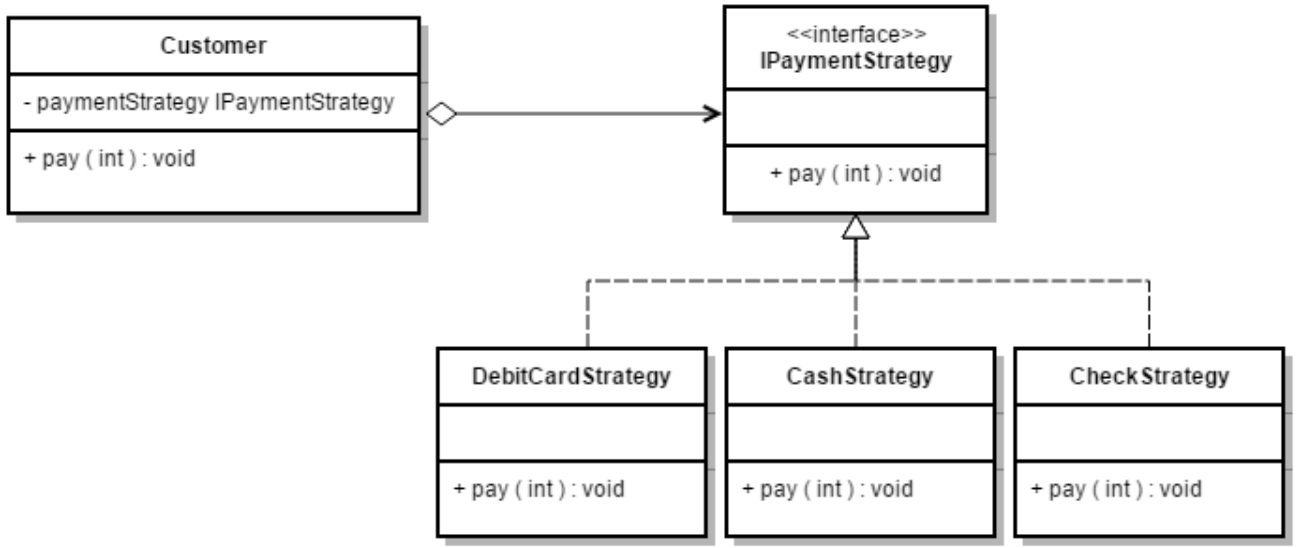



Diagrama UML

```
class Customer
{
    private IList<double> drinks;

    // Get/Set Strategy
    public IBillingStrategy Strategy { get; set; }

    public Customer(IBillingStrategy strategy)
    {
        this.drinks = new List<double>();
        this.Strategy = strategy;
    }

    public void Add(double price, int quantity)
    {
        this.drinks.Add(this.Strategy.GetActPrice(price * quantity));
    }

    // Payment of bill
    public void PrintBill()
    {
        double sum = 0;
        foreach (double i in this.drinks)
        {
            sum += i;
        }
        Console.WriteLine("Total due: " + sum);
        this.drinks.Clear();
    }
}
```

```
interface IBillingStrategy
{
    double GetActPrice(double rawPrice);
}

// Normal billing strategy (unchanged price)
class NormalStrategy : IBillingStrategy
{
    public double GetActPrice(double rawPrice)
    {
        return rawPrice;
    }
}

// Strategy for Happy hour (50% discount)
class HappyHourStrategy : IBillingStrategy
{
    public double GetActPrice(double rawPrice)
    {
        return rawPrice * 0.5;
    }
}
```

Implementación