

# REQUIREMENTS VERIFICATION & VALIDATION

☰ Objetivo

☰ Introducción

☰ Acceptance tests

?

Actividad 1

☰ Prototyping

?

Actividad 2

☰ Review & Retrospective

?

Actividad 3

☰ Resumen

☰ Bibliografía

☰ PDF descargable

## OBJETIVO

---

Al finalizar la unidad, el estudiante comunica las especificaciones de los requisitos a partir de las metodologías actuales de desarrollo de software, de acuerdo a las necesidades de los involucrados de manera clara y fluida.



## INTRODUCCIÓN

---

## VERIFICATION

Se refiere al conjunto de tareas que aseguren que el software implementa de manera correcta una función específica.

## VALIDATION

Se refiere al conjunto de tareas que aseguren que el software que se ha construido tiene una relación clara con los requisitos del cliente.

---

**Si los requisitos no se validan, los errores presentes en las definiciones de requisitos podrían propagarse a las etapas sucesivas, llevando a mayor cantidad de modificaciones y retrabajo.**

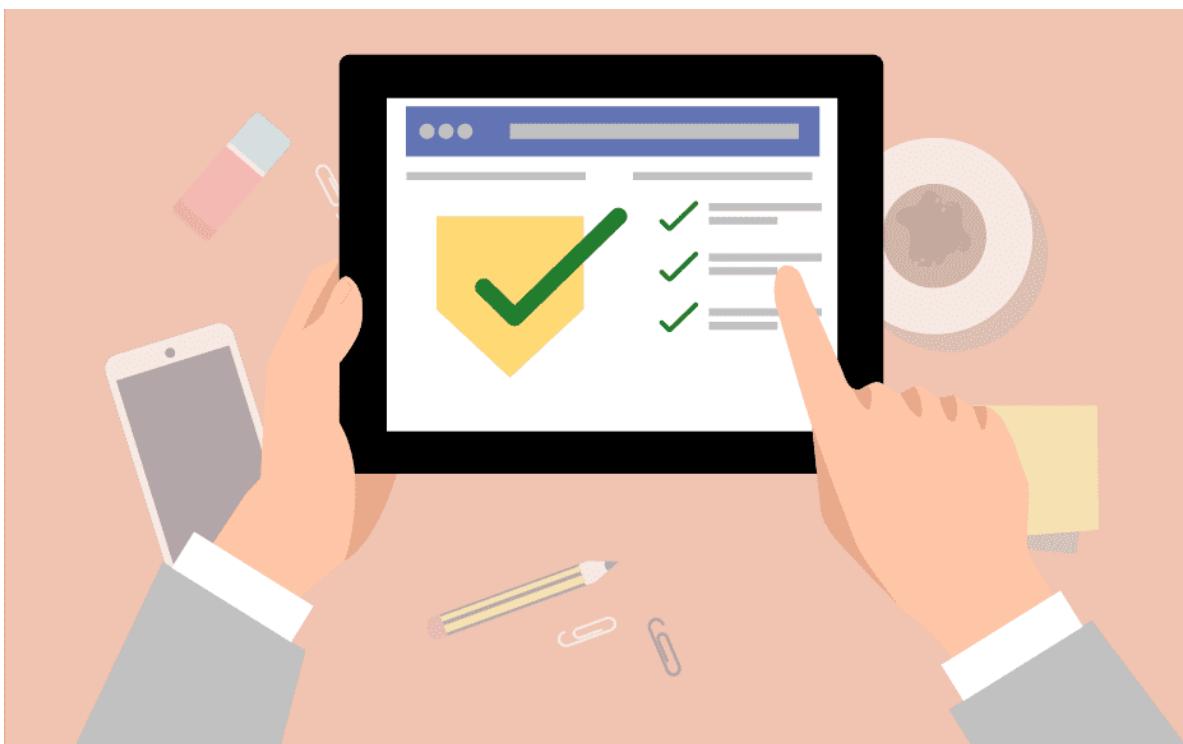
Los requisitos deberían:

- Ser consistentes con los demás requisitos; es decir, que no debería darse que dos requisitos estén en conflicto.
- Estar completos en todo sentido.
- Ser alcanzables en términos prácticos.



## REQUIREMENTS VALIDATION

Como se vio anteriormente, es el proceso de comprobar que los requisitos definidos para el desarrollo en realidad definen el sistema que el cliente desea. Para comprobar aspectos relacionados a los requisitos, se realiza la validación de requisitos.



Normalmente se aplica la validación de requisitos para comprobar errores en la fase inicial de desarrollo, dado que el error podría incrementar considerablemente el retrabajo cuando se detecta en etapas posteriores del desarrollo.

## REQUIREMENTS VALIDATION PROCESS

En el proceso de *requirements validation* se realizan diferentes tipos de pruebas para comprobar los requisitos especificados. Estas incluyen:

- Completeness checks (comprobar si la especificación está completa)
- Consistency checks (comprobar si es consistente)
- Validity checks (comprobar si es válido)
- Realism checks (comprobar si es realista)
- Ambiguity checks (comprobar si hay ambigüedad)
- Verifiability (comprobar si es verificable)

## REQUIREMENTS VALIDATION OUTPUT

Es la lista de problemas y las recomendaciones de acción sobre los problemas detectados.

Haz clic en cada tarjeta para ver la información.

## LIST OF PROBLEMS



Son los problemas detectados durante el proceso de Requirements Validation.

## LIST OF AGREED ACTIONS



Son los acuerdos tomados sobre acciones a realizar en relación a los problemas detectados.

# TECHNIQUES

Hay varias técnicas que se utilizan de forma individual o en conjunto con otras para comprobar de forma completa o parcial un sistema.

Estas técnicas son:

## TEST CASE GENERATION

Los requisitos deberían poder probarse, para ello, los test revelan errores presentes en los requisitos. Si se torna difícil o imposible diseñar una prueba para un requisito, es de esperarse que la implementación será difícil y amerita revisión.

The screenshot shows a software interface for creating test cases. At the top, there's a toolbar with icons for 'X' (close), '+', and three colored circles (green, yellow, red). Below the toolbar is a navigation bar with icons for back/forward, home, and refresh, followed by a search bar and a star icon.

The main area has several tabs at the top: 'Date run', 'Time run', 'Tester', 'Pass/Fail', and 'Summary of Failure'. The 'Pass/Fail' tab is currently selected, indicated by a blue background.

Below the tabs, there are several input fields and sections:

- Assumptions:** A text area containing the instruction: **Here you write your assumptions; the pre-conditions; or the initial state.**
- System Requirement Covered:** A text area containing the instruction: **The number of the function that's being tested.**
- Exclusions:** A text area containing the instruction: **Exclusions, like any un-desired values.**
- Setup: Steps taken:** A text area containing the instruction: **Write the steps that should be taken by the user for this function.**
- User Action:** A text area containing the instruction: **What are the inputs**
- Value(s):** A text area containing the instruction: **List all the values that will be entered by the user**
- Expected Results:** A text area containing the instruction: **The expected output**
- Pass/Fail/Untested:** A text area where the status of the test case is recorded.

## PROTOTYPING

En esta técnica de validación, el prototipo del sistema (una aproximación con capacidad de interacción que permite tener una idea de lo que significaría el sistema real) se presenta a los usuarios finales o clientes, a fin de que experimenten con el modelo actual y comprueben que satisface sus necesidades.

Este tipo de modelo se utiliza para recolectar *feedback* sobre los requisitos del usuario.

## REQUIREMENTS REVIEWS

En esta aproximación, el SRS o Backlog es revisado con atención por miembros del negocio y miembros del equipo de desarrollo, con el fin de detectar errores o ambigüedades.

En un enfoque agile, esto se lleva a cabo en las Backlog Refinement Meetings.

## AUTOMATED CONSISTENCY ANALYSIS

Este *approach* se utiliza para la detección automática de errores, como no determinismo, escenarios no cubiertos, errores de tipos definiciones circulares a nivel de las especificaciones de requisitos.

El requisito se estructura usando notación formal para aplicar una herramienta de comprobación de consistencia del sistema, para que se identifiquen inconsistencias y se realicen las acciones correctivas.

Aquí se hace uso de herramientas para análisis de lenguaje natural (NL) como QuARS (<http://quars.isti.cnr.it>).

#### WALK-THROUGH

No tiene un procedimiento definido o roles específicos. En este *approach* se realizan las siguientes actividades:

- Comprobar lo antes posible si la idea es factible o no.
- Obtener opiniones y sugerencias de otras personas.
- Comprobar si otros aprueban la especificación de requisitos y lograr acuerdos.

## ACCEPTANCE TESTS



## SCENARIOS

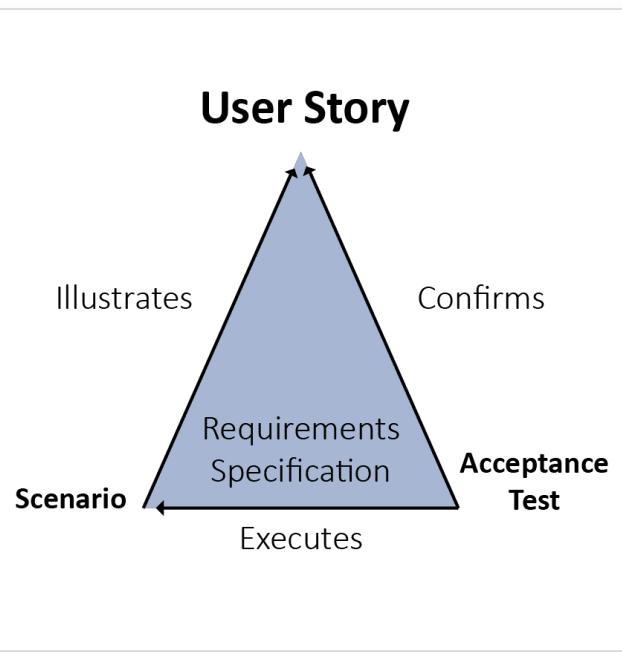
Anteriormente se vio que los escenarios ayudan en el monitoreo de la conversación y a confirmar las expectativas. Se entiende que los escenarios expresan los *success criteria* de lo que se entrega en un sprint.

**¿Cómo confirmar que se cumple con los success criteria?**

**Los Acceptance Tests** pueden confirmar que se cumple con los *success criteria*.

## EVOLUCIONANDO SCENARIOS HACIA ACCEPTANCE TEST

Si se han redactado los *scenarios* correctamente, elaborar *acceptance tests* a partir de ellos puede realizarse con mínimos cambios.



Un *acceptance test* no es otra cosa que una copia de un *scenario* en un formato adecuado para su ejecución en una computadora.

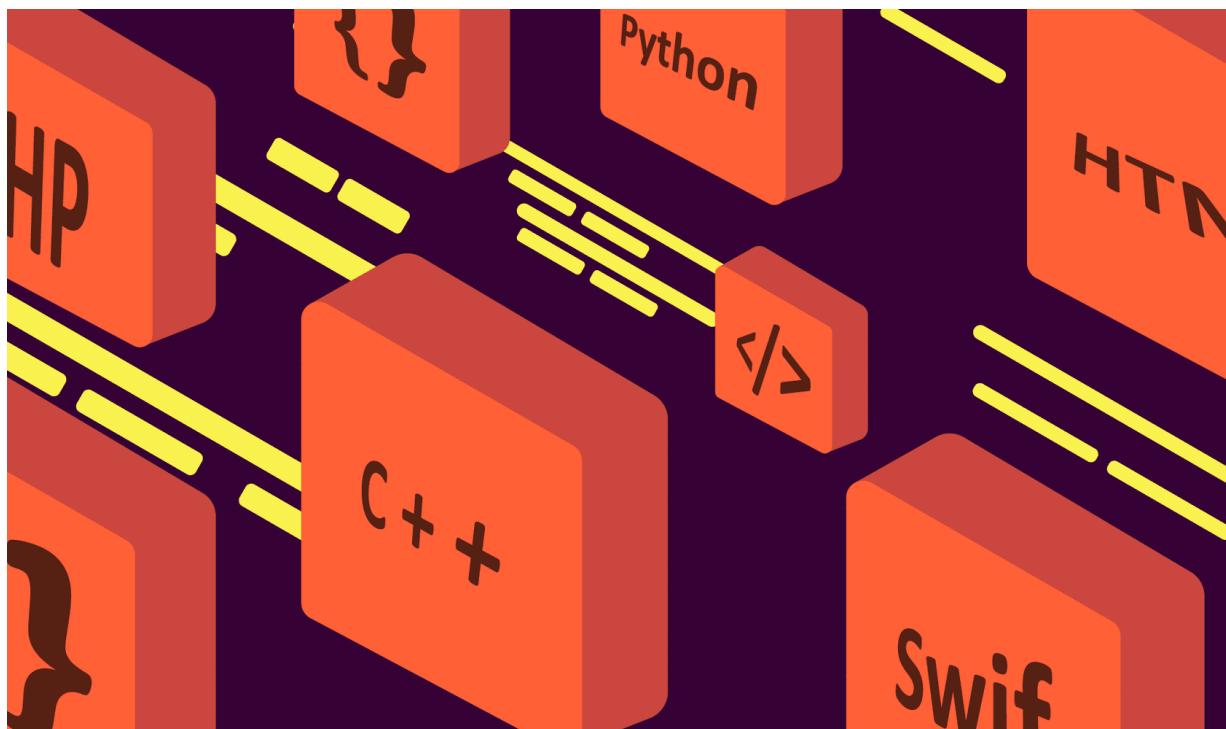
Debe lucir como el *scenario* original, incluso considerando las

## TRADUCIENDO SCENARIOS HACIA ACCEPTANCE TEST



El *tester* desarrolla un *acceptance test* siguiendo una secuencia específica de *steps* según indican los *constraints*.

---



Estos *constraints* existen para asegurar que los supuestos técnicos seleccionados durante el diseño no hagan que la implementación difiera de la intención original del *scenario*.

---



El *acceptance test* se implementa con programación.

---



A fin de evitar falsos supuestos, el proceso de traducción no debería requerir supuestos. Para ello, se apoya en un *domain-specific language* (DSL).

---

## TRANSPONDIENDO USANDO UN INTERNAL DSL



“Martin Fowler define un DSL como un lenguaje de programación con un vocabulario limitado enfocado en un dominio en particular.”

El DSL se incrusta en el lenguaje de programación que el equipo de desarrollo utiliza. Una solución típica para esto es utilizar un framework de BDD, como Cucumber o SpecFlow.



Según se ha visto, un scenario se expresa de la siguiente manera:

**Given** an empty shopping cart is created

**And** a monthly student pass is added to shopping cart

**When** the buyer checks out the shopping cart

**Then** a 76 dollar sale occurs.

A continuación, se presentan algunos ejemplos de traducción de escenarios y sus pasos utilizando lenguajes de programación como Ruby, Java y C#.

```
Given /^an empty shopping cart is created $/ do
  ...
End
Given /^ a (.*) is added to shopping cart $/ do |pass|
  ...
End
When /^the buyer checks out the shopping cart $/ do
  ...
End
Then /^a ([0-9]*) dollar sale occurs $/ do |amount|
  ...
End
```

Traduciendo a Ruby.

```
@Given ("^an empty shopping cart is created $")
public void GivenAnEmptyShoppingCartIsCreated ()
{
    ...
}
@Given ("^a (.*)is added to shopping cart$")
public void GivenA_IsAddedToShoppingCart (string pass)
{
    ...
}
@When ("^the buyer checks out the shopping cart$")
public void WhenTheBuyerChecksOutTheShoppingCart ()
{
    ...
}
@Then ("^a ([0-9]*) dollar sale occurs$")
public void ThenA_DollarSaleOccurs (money amount)
{
    ...
}
```

Traduciendo a Java.

```
new Scenario("Feature: Shopping Cart")
    .Given(AnEmptyShoppingCartIsCreated )
    .Given(APassIsAddedToShoppingCart, pass: "student monthly
        pass" )
    .When(TheBuyerChecksOutTheShoppingCart)
    .Then(ANumberDollarSaleOccurs, number: 76)
    .Execute();
```

Traduciendo a C# con una fluent interface.

## ESCRIBIENDO UN ACCEPTANCE TEST

Para la implementación de los Acceptance Test, existen herramientas que permiten la implementación de pruebas de aceptación automatizadas, como SpecFlow o StoryQ en C#.

C# CON SPECFLOW

C# CON STORYQ

C++ Y CUCUMBER-CPP

Redactando un acceptance test usando C# con SpecFlow.

```
[Binding] ←
public class ChecksOutPassTest
{
    [Given ("an empty shopping cart is created")]
    public void AnEmptyShoppingCartIsCreated()
    {

    }

    [Given ("a (.*) is added to shopping cart")]
    public void APassIsAddedToShoppingCart(string pass)
    {

    }

    [When ("the buyer checks out the shopping cart")]
    public void TheBuyerChecksOutTheShoppingCart()
    {

    }

    [Then ("a (.*) dollar sale occurs")]
    public void ANumberDollarSaleOccurs(int number)
    {
    }
}
```

1 Embed the scenario in a SpecFlow class

2 Given-When-Then steps

C# CON SPECFLOW

C# CON STORYQ

C++ Y CUCUMBER-CPP

Implementando el acceptance test usando C# con StoryQ.

```

[TestClass]
public class ChecksOutPassTest
{
    [TestMethod]
    public void ChecksOutPassScenario()
    {
        new Story("Transit Fare")
            .InOrderTo("go to school and get around")
            .AsA("student")
            .IWantTo("buy a transit fare")
            .WithScenario("Checkout pass") ← 1 Scenario express using a DSL
                .Given(AnEmptyShoppingCartIsCreated)
                .And(APassIsAddedToShoppingCart, pass: "student monthly pass")
                .When(TheBuyerChecksOutTheShoppingCart)
                .Then(ANumberDollarSaleOccurs, number: 76)
                .ExecuteWithReport(MethodBase.GetCurrentMethod());
    }

    void AnEmptyShoppingCartIsCreated ()
    {
    }
    void APassIsAddedToShoppingCart (string pass) ← 2 Given-When-Then steps
    {
    }
    void TheBuyerChecksOutTheShoppingCart()
    {
    }
    void ANumberDollarSaleOccurs(int number)
    {
    }
}

```

C# CON SPECFLOW

C# CON STORYQ

C++ Y CUCUMBER-CPP

Implementando el acceptance test usando C++ y Cucumber-Cpp.

```

#include <CppSpec/CppSpec.h>
#include <cucumber-cpp/defs.hpp>

struct CalcCtx {
    Calculator calc;
    double result;
};

using cucumber::ScenarioScope;

GIVEN("^I have entered (\\d+) into the calculator$") {
    REGEX_PARAM(double, n);
    ScenarioScope<CalcCtx> context;
    context->calc.push(n);
}

WHEN("^I press add") {
    ScenarioScope<CalcCtx> context;
    context->result = context->calc.add();
}

WHEN("^I press divide") {
    ScenarioScope<CalcCtx> context;
    context->result = context->calc.divide();
}

THEN("^the result should be (.*) on the screen$") {
    REGEX_PARAM(double, expected);
    ScenarioScope<CalcCtx> context;
    specify(context->result, should.equal(expected));
}

```

## MEJORANDO SCENARIOS CON TEST RESULTS

Muchas de las herramientas, como el caso de Cucumber o SpecFlow, permiten la generación de reportes que incluyen esquemas de colores, como:

- Amarillo para pendiente

- Verde para aprobado

- Rojo para fallido

## Feature: Shopping Cart

### Scenario

**Given** the buyer is logged as a student  
**When** the buyer requests the list of transit fares

**Then**

<i>Id</i>	<i>Name</i>	<i>Price</i>
002	Student monthly pass	76
100	One day pass	6
202	Student booklet of 10 single tickets	15

**Pending Implementation**

### Scenario

**Given** an empty shopping cart is created  
And a monthly student pass is added to shopping cart  
**When** the buyer checks out the shopping cart  
**Then** a 76 dollar sale occurs

**Passed**

### Scenario

**Given** an empty shopping cart is created  
And a monthly student pass is added to shopping cart  
**When** the buyer removes all items from shopping cart  
**Then** all items are removed from shopping cart

**Fail**

Aplicando esquemas de colores (Yellow, Green, Red).

## INICIANDO CON ACCEPTANCE TESTS

En etapas iniciales puede esbozar *acceptance tests*:

- Redactando las pruebas para los *scenarios* como pseudocódigo basado en el DSL con Given-When-Then.
- Especificando los datos de prueba para cada *scenario*.

## ACTIVIDAD 1

---

Lee los siguientes enunciados y marca la respuesta correcta:

---

**01/03**

La \_\_\_\_\_ se refiere al conjunto de tareas que aseguren que el software que se ha construido tiene una relación clara con los requisitos del cliente.

---

validation

verification

## 02/03

Asocia las técnicas con su característica:

---

☰ Prototyping

Revelan errores presentes en los requisitos.

☰ Test case generation

El prototipo del sistema se presenta a los usuarios finales o clientes.

☰ Requirements Reviews

Es revisado por miembros del negocio y miembros del equipo de desarrollo.

☰ Automated Consistency Analysis

Se hace uso de herramientas para análisis de lenguaje natural.

**03/03**

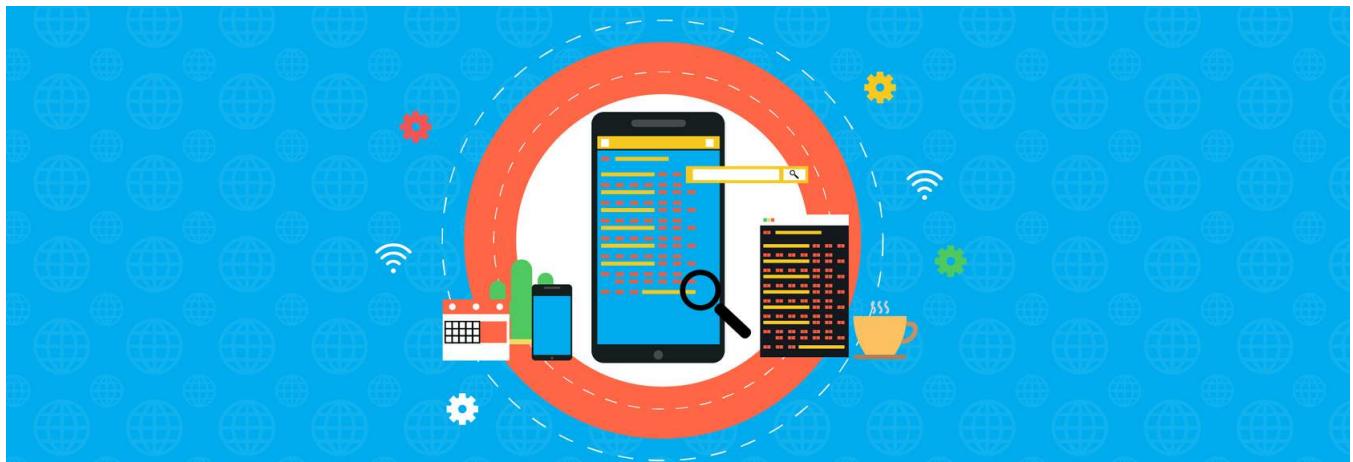
Marca las alternativas correctas respecto a un acceptan test.

---

- Es una copia de un scenario en un formato adecuado para su ejecución en una computadora.
- Ayudan en el monitoreo de la conversación y a confirmar las expectativas.
- Debe lucir como el scenario original.

## PROTOTYPING

---

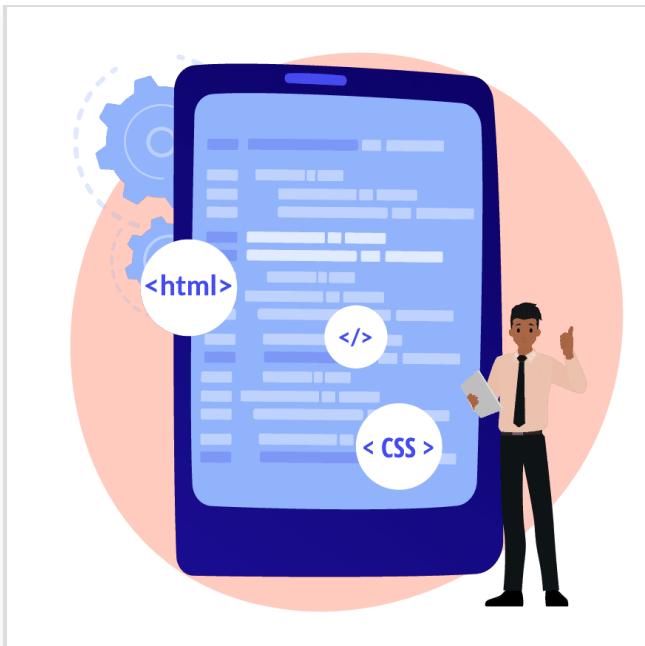


## PROTOTYPE

Muestra, modelo o versión de un producto construido para probar un concepto o proceso.

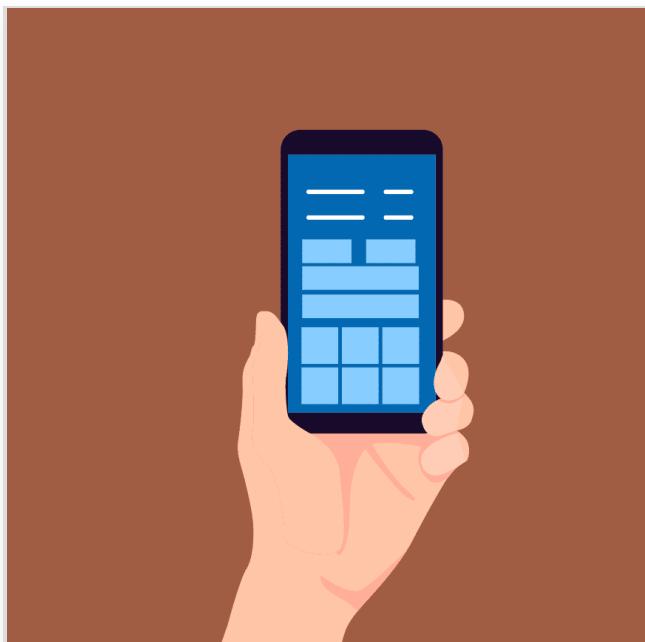
**i** Deriva del griego *prototypon*: “forma primitiva”.

## CATEGORÍAS



### PROOF-OF-PRINCIPLE PROTOTYPE

Sirve para verificar aspectos funcionales específicos del diseño propuesto, pero es usualmente un subconjunto de la funcionalidad del producto final.



### WORKING PROTOTYPE

Representa toda o casi toda la funcionalidad del producto final.

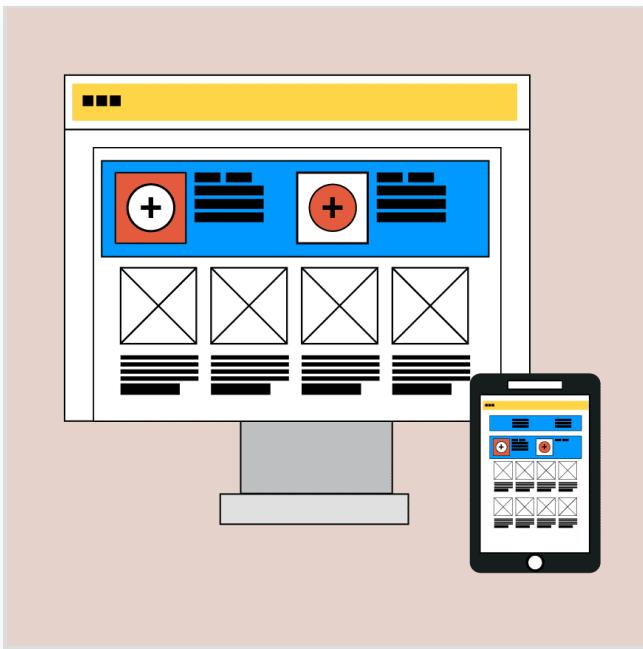


### VISUAL PROTOTYPE

Representa el tamaño y apariencia, más no la funcionalidad del diseño propuesto.

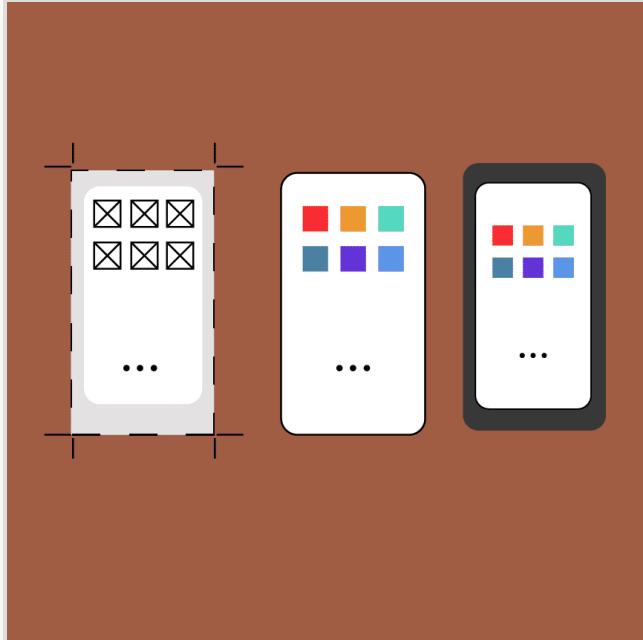


Planning



### USER EXPERIENCE PROTOTYPE

Representa al producto en apariencia y funcionalidad, lo suficiente como para utilizarlo en *user research*.



### FUNCTIONAL PROTOTYPE

Representa el diseño propuesto en funcionalidad y apariencia, aunque puede elaborarse con técnicas, materiales y escalas diferentes.



Prototyping



### PAPER PROTOTYPE

Representación impresa o a mano de la interfaz de un producto de software (útil en pruebas preliminares de diseño y en software walk-through).

En UX no se aplica el término Paper Prototype, se utiliza en su reemplazo términos según el grado de fidelidad de la representación: Wireframe (low-fidelity o low-res) y Mockup (high-fidelity o high-res).

## RAPID PROTOTYPING

- 1 Aplica una aproximación iterativa a la etapa de diseño de una app o website.
- 2 El objetivo es mejorar lo más rápido posible el diseño, utilizando prototipos actualizados con regularidad en ciclos cortos.
- 3 Busca reducir costos y tiempo resolviendo problemas comunes de diseño antes de iniciar el desarrollo, centrándose en las necesidades de los usuarios.

4

Se originó en *manufacturing*.

5

Normalmente asociado con RAD (Rapid Application Development).

## RAPID PROTOTYPING PROCESS

### STEPS



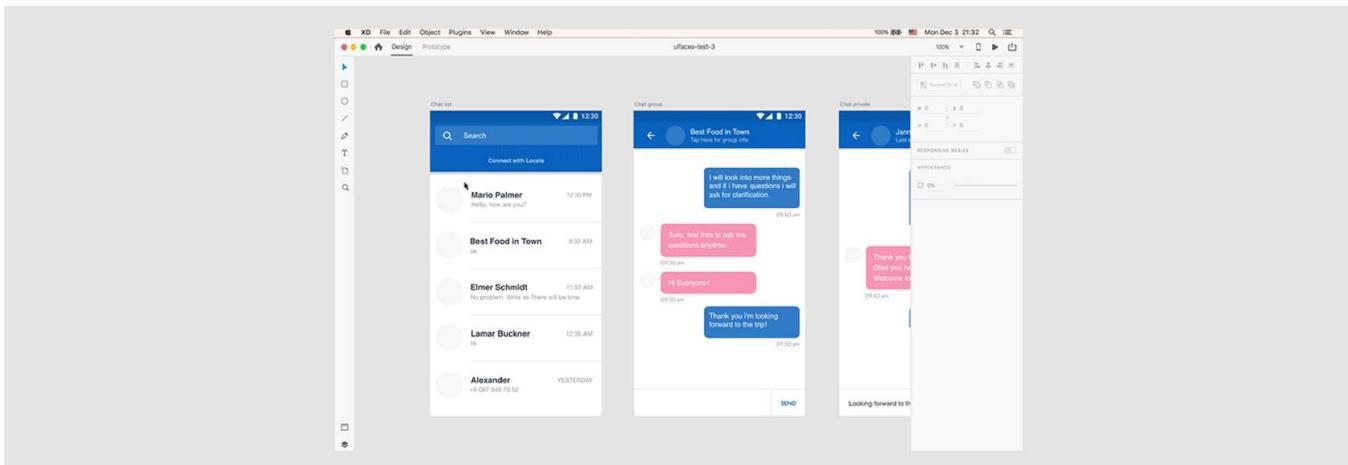
## RAPID PROTOTYPING AND AGILE METHODS

Rapid Prototyping y Agile implican mejoras incrementales en múltiples iteraciones, pero no son lo mismo.

*Rapid Prototyping* implica iterar sobre la fase de *design and planning*; mientras que, en *Agile*, el proceso iterativo e incremental es durante la fase de *development*. En Agile se crea un MVP (Minimum Viable Product) el cual se va probando y mejorando en múltiples ciclos.

Un MVP puede aparentar ser un prototipo en su inicio, pero en realidad evolucionará hasta convertirse en el producto final. Un prototipo ayuda a aclarar requisitos y el diseño, pero finalmente será descartado.

Algunas herramientas para elaborar prototipos son:



- Adobe XD
- UXPin
- JustInMind
- Sketch App

ⓘ Enlace: <https://www.adobe.com/products/xd.html>

## ACTIVIDAD 2

---

Lee los siguientes enunciados y marca la respuesta correcta:

---

**01/03**

¿Cuál es la categoría de Prototype que representa toda o casi toda la funcionalidad del producto final?

---

- Visual Prototype
- Functional Prototype
- Working Prototype

**02/03**

La categoría de prototype denominada \_\_\_\_\_ representa al producto en apariencia y funcionalidad, lo suficiente como para utilizarlo en user research.

---

- User Experience Prototype
- Paper Prototype
- Proof-of-Principle Prototype

### 03/03

¿Qué es el Paper Prototype?

---

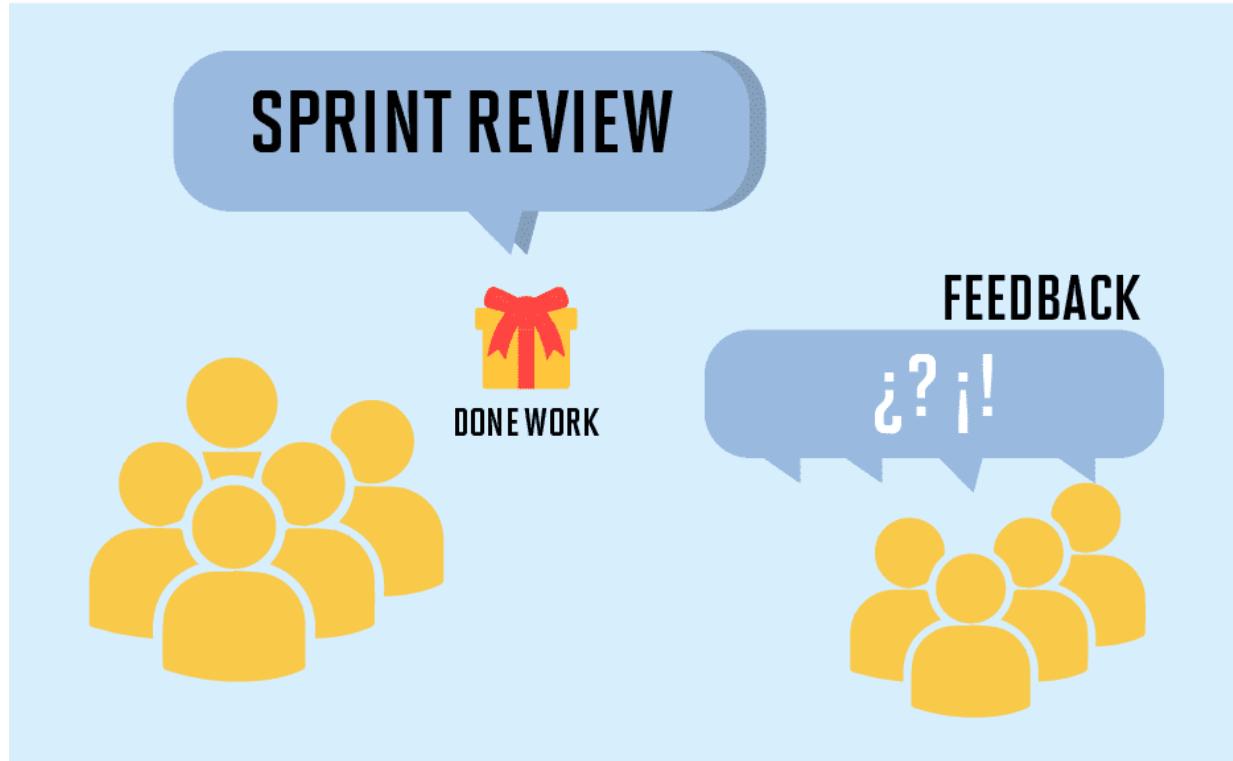
- Representación del diseño propuesto en funcionalidad y apariencia, con técnicas, materiales y escalas diferentes.
- Representación del tamaño y apariencia, más no la funcionalidad del diseño propuesto.
- Representación impresa o a mano de la interfaz de un producto de software.

## REVIEW & RETROSPECTIVE

---



### SPRINT REVIEW



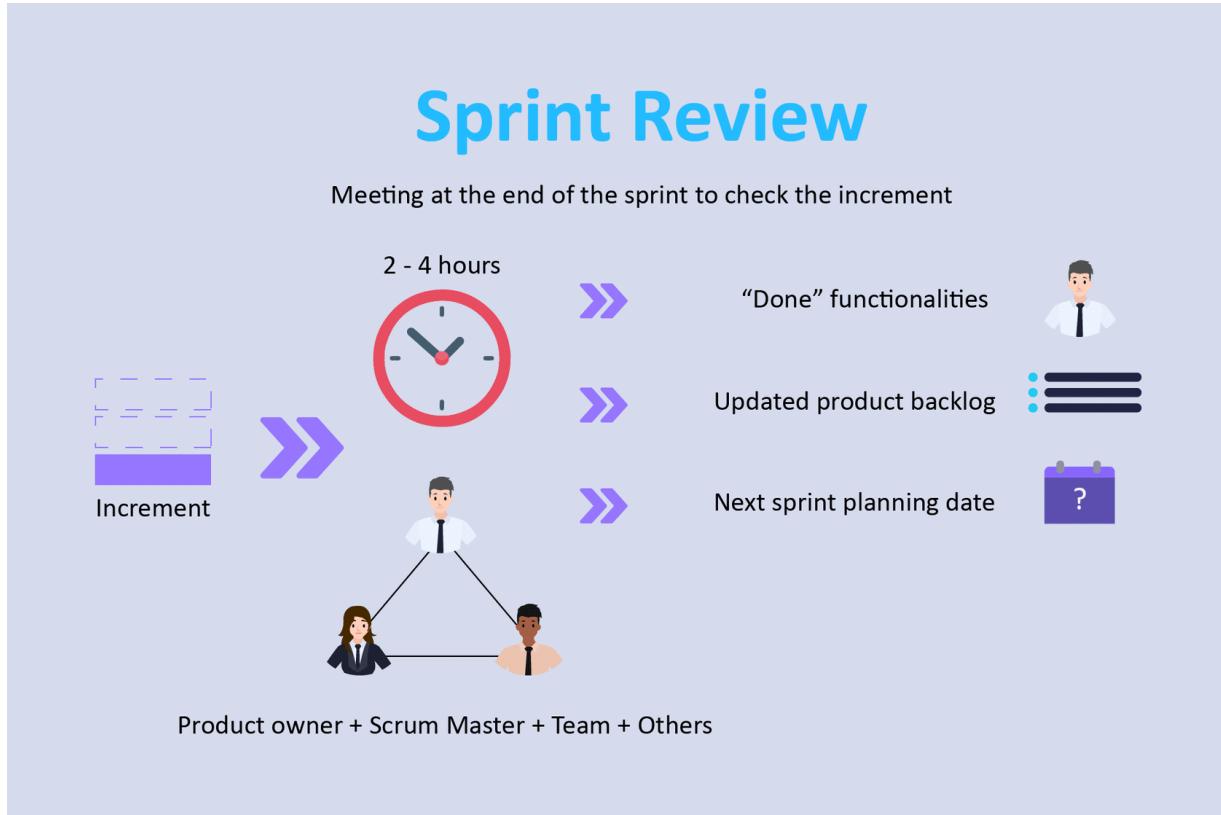
Se da el último día del sprint. Deberían asistir el product owner, Scrum Master, development team y los stakeholders apropiados; según los features que se vaya a entregar en el sprint.

Dura un máximo de 4 horas para un 4-week sprint.

**Input:** El team muestra los items de product backlog que cumplen con los criterios de DoD (Definition-of-Done). No se muestra trabajo en progreso.

El product owner toma en consideración el feedback generado y puede hacer cambios en el product backlog.

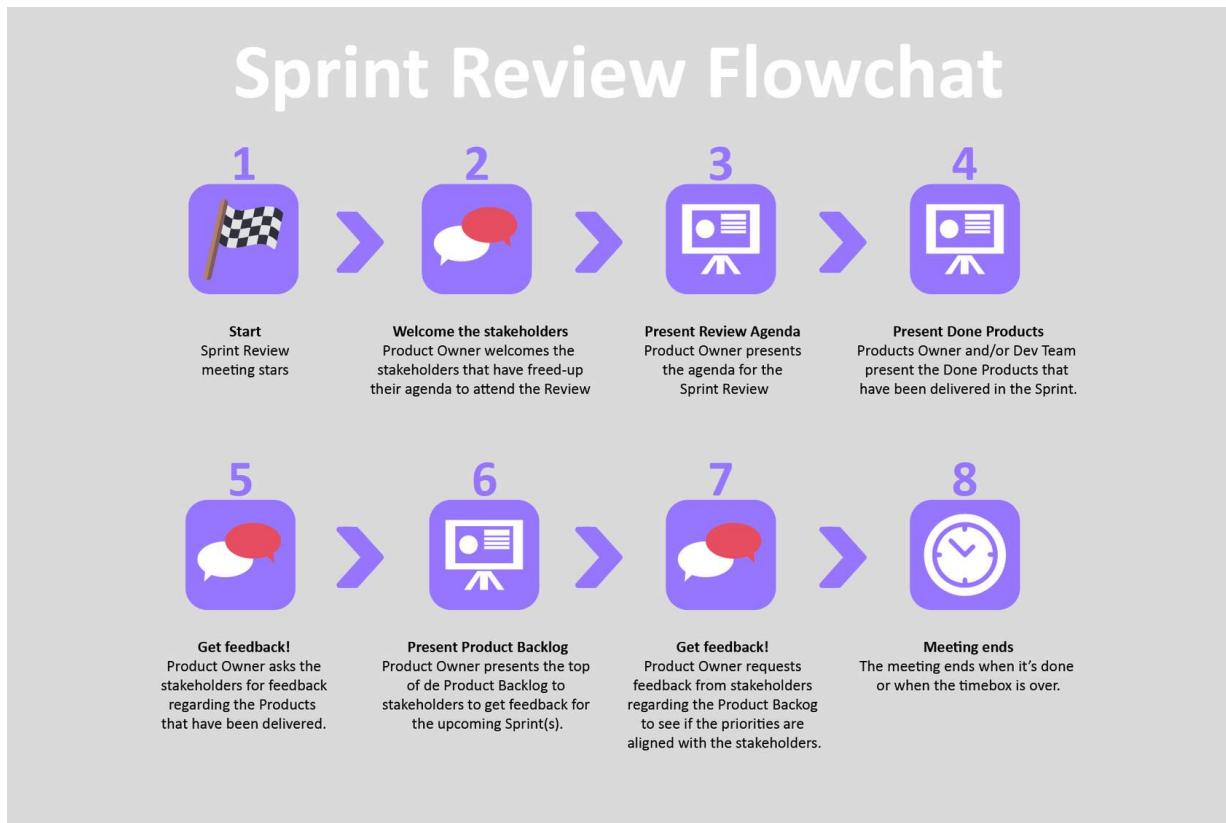
**Output:** Revised Product Backlog.



## Agenda

- 1 Bienvenida a los participantes y establecer reglas o expectativas para el Sprint Review.
- 2 Establecer qué (y qué no) se demostrará.
- 3 Demo de nueva funcionalidad.
- 4 Discutir eventos o problemas clave ocurridos durante el sprint.

- 5 Presentar Product Backlog Items que vendrán.
- 6 Concluir la reunión.
- 7 After meeting. No parte de la agenda, pero debería actualizarse en la herramienta de control.

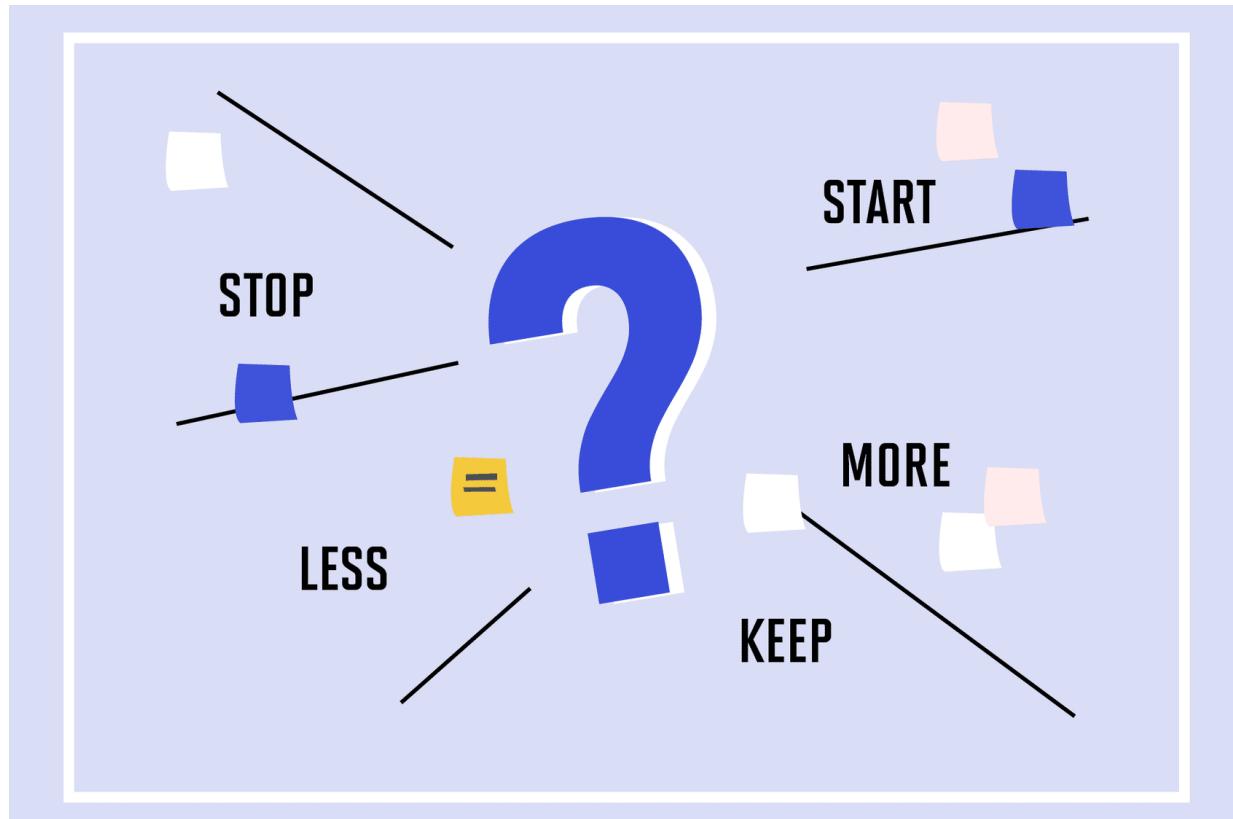


Este gráfico resume el flujo que debería seguir un Sprint Review, cubriendo este la presentación de software implementado que estaba acordado para este sprint, así como la revisión del product backlog sobre el que se planificaría el siguiente sprint.

## SPRINT RETROSPECTIVE

Esta ceremonia permite que los team members consideren cómo mejorar en su forma de trabajo. Pueden modificar aspectos de cómo aplican Scrum, como la duración de sus sprints.

Debería acudir todo el team (development team, Scrum Master y product owner). La participación del product owner busca mantener transparencia en el equipo.



**Input:** No hay input formal.

**Output:** Lista de cambios a implementar en la forma de trabajo del equipo.

No debe durar más de 3 horas, aunque tiende a durar 1 hora.

# Sprint Retrospective

Meeting after Sprint Review to review processes

- What went well?
- What could be improved? ➤
- How can we improve it?



Product owner + Scrum team



➤ Self - analysis on how to work



➤ Problem analysis and improved aspects



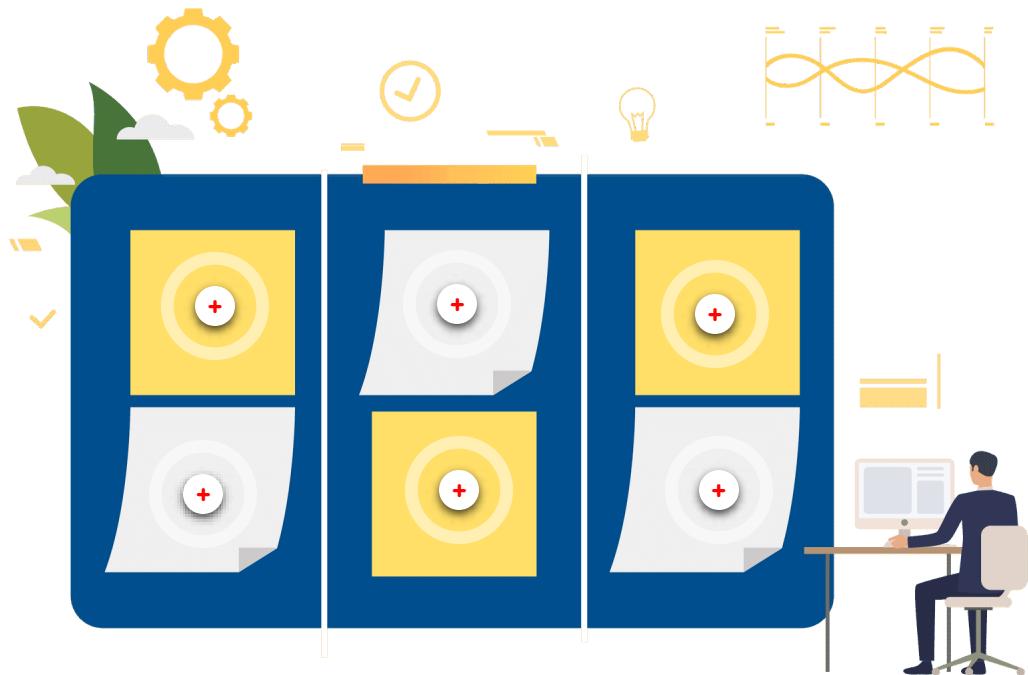
➤ Framework improvements

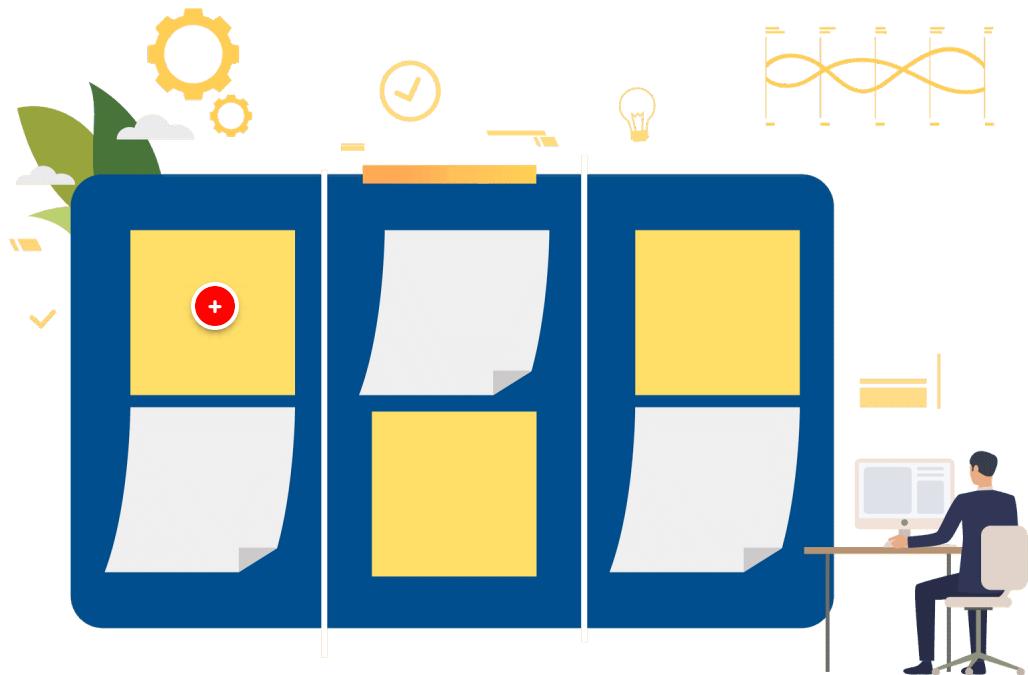


## PRODUCT BACKLOG REFINEMENT

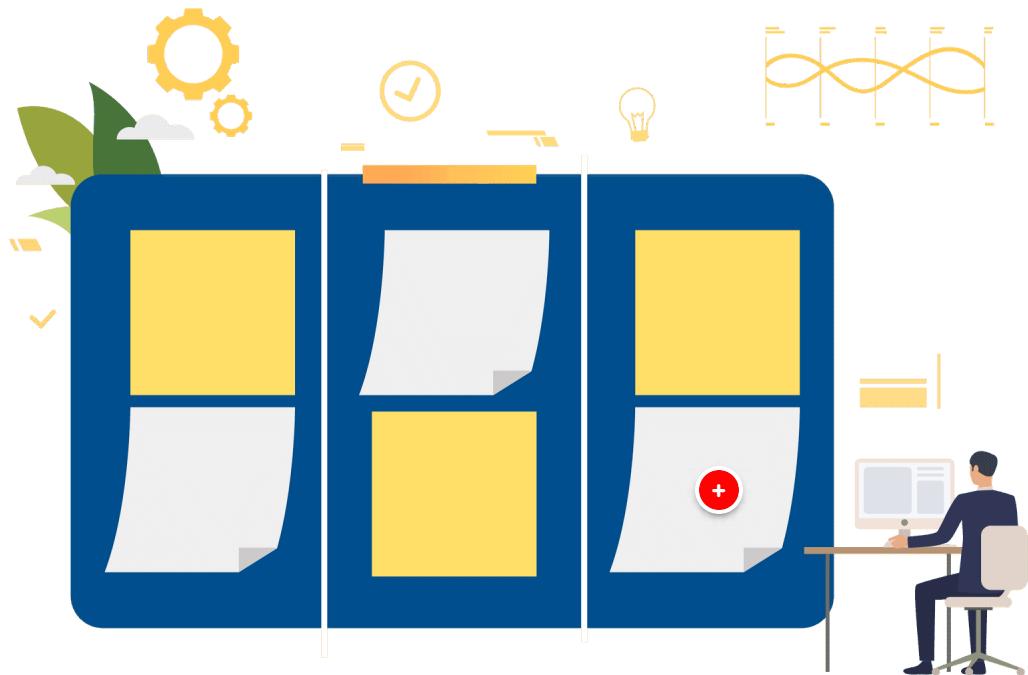
Busca asegurarse que los product backlog items del tope estén listos para el siguiente sprint. No es obligatoria una ceremonia formal, aunque usualmente se realizan reuniones de refinamiento; una en el sprint o una por semana en algunos casos, tratando de no usar más del 10 % del tiempo disponible del equipo.

Incluye:

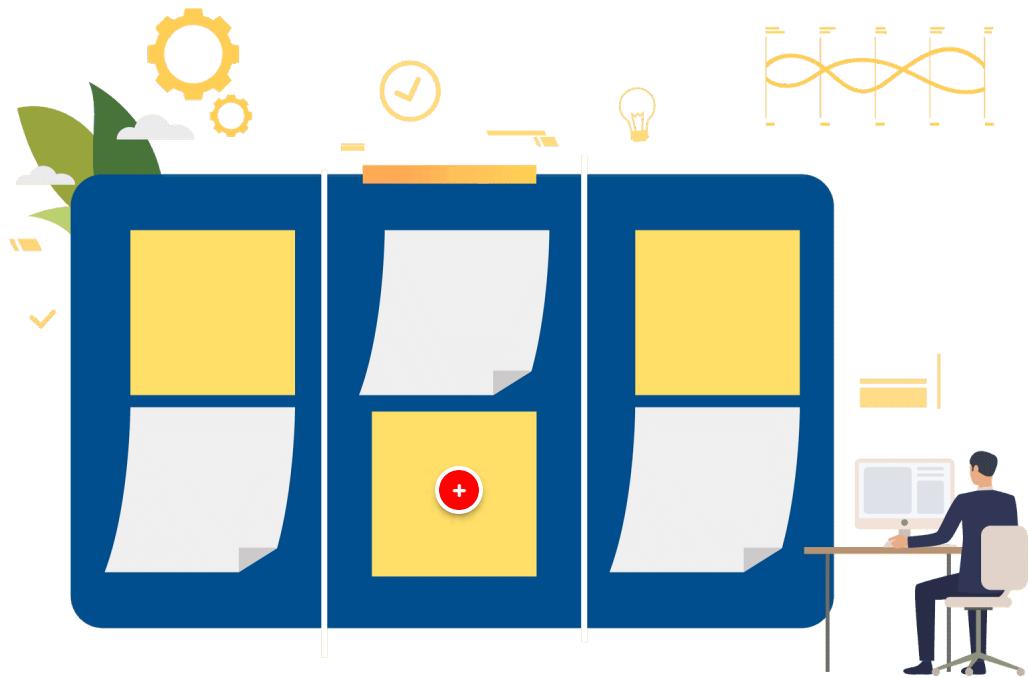




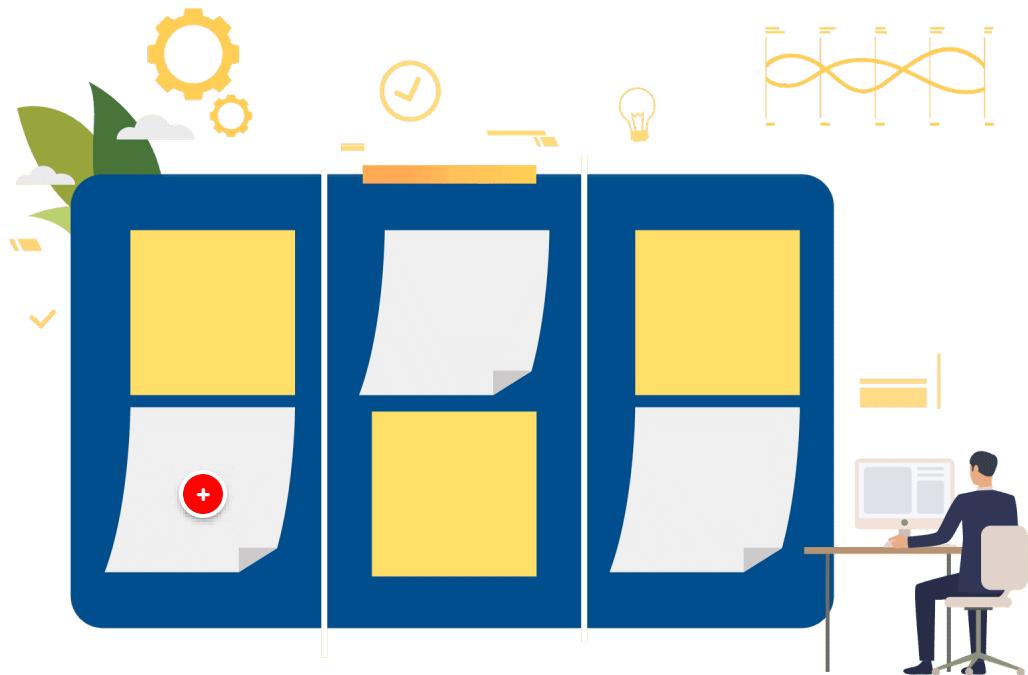
Agregar detalle a ítems existentes.



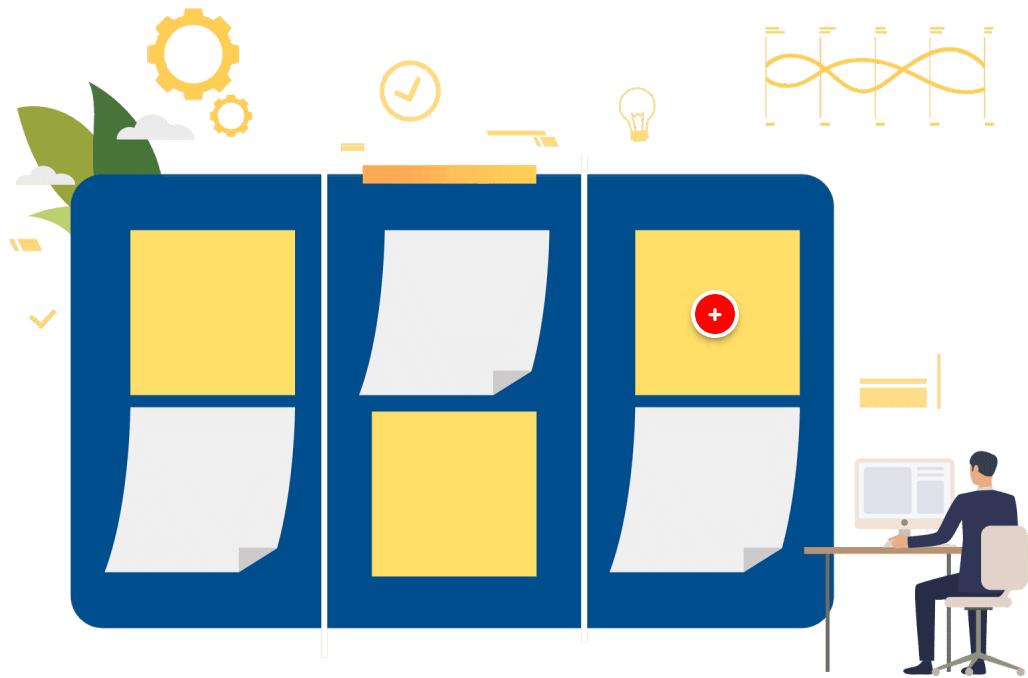
**Crear nuevos ítems.**



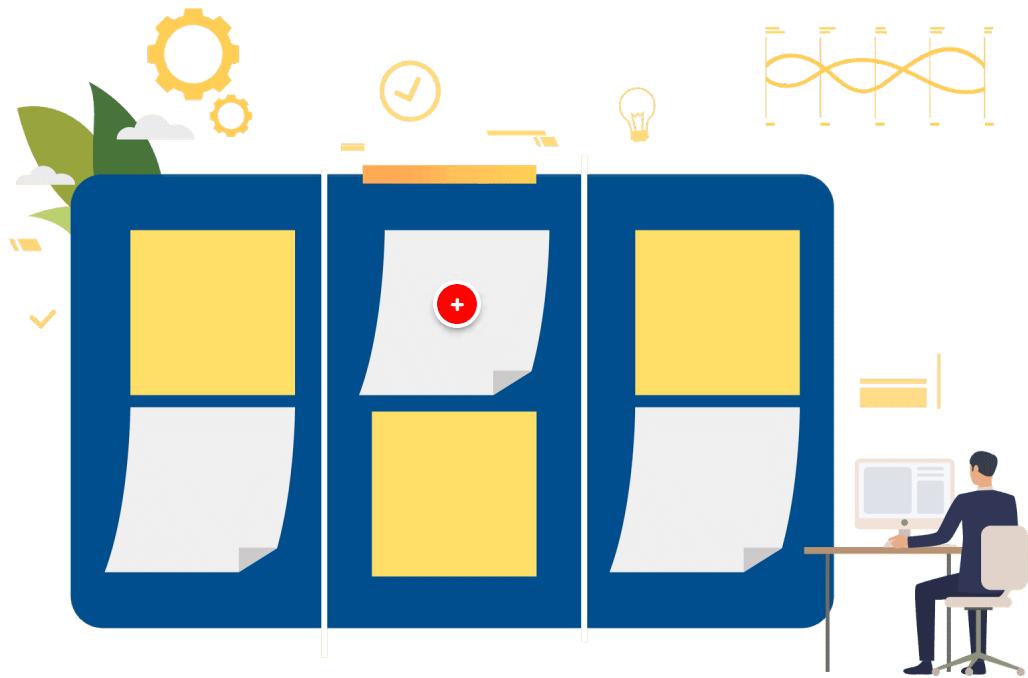
Descomponer items para que encajen mejor en un sprint.



Ajustar prioridades.



**Eliminar items.**



## Estimar.

En los siguientes gráficos se resume el flujo que debería seguir el Refinement Meeting, desde la preparación a cargo del Product Owner hasta la actualización del backlog, dejándolo listo para el Sprint planning.

## Refinement Meeting Flowchart | Before Meeting



**Review the PBL**  
Product Owner reviews the Product Backlog

**Select Items to Refine**  
Product Owner selects a set of items to be refined

**Add the known details**  
Product Owner Adds the known details (Why + What + Acc. Criteria)

**Prepare for Refinement**  
Product Owner prepares to present the items at the refinement meeting

## Refinement Meeting Flowchart | During Meeting



**Start**  
The Refinement meeting starts

**Present Backlog Items**  
Product Owner presents the backlog items to be refined

**Present the first item**  
Product Owner presents the Why, What and Acc. Criteria for the first Item

**Time-boxed discussion**  
The Dev Team gets to ask questions and discuss about the Backlog Item (10 min)



**Ready for Estimation?**  
Is the item ready to be estimated? If yes, continue to 6. If no, go back to 4

**Planning Poker**  
Apply Planning Poker to estimate the size of the Backlog item

**Update the Backlog**  
Update the Backlog Item with the new information gained in the meeting

**Time left?**  
If there's time left, continue with the next item, otherwise continue next time!



## ACTIVIDAD 3

---

Lee los siguientes enunciados y marca la respuesta correcta:

---

**01/03**

Selecciona lo correcto respecto al Sprint Review.

---

- Permite que los team members consideren cómo mejorar en su forma de trabajo.
- Se da el último día del sprint.
- En el Input, el team muestra los items de product backlog que cumplen con los criterios de DoD.

**02/03**

La ceremonia del \_\_\_\_\_ permite que los team members consideren cómo mejorar en su forma de trabajo.

---

- Sprint Review
- Sprint Retrospective
- Product Backlog Refinement

**03/03**

¿Qué acciones se llevan a cabo en el Product Backlog Refinement?

---

- Agregar detalles a items existentes
- Debatir items
- Eliminar items
- Ajustar prioridades

## RESUMEN

---



- Acceptance Tests
- Prototyping
- Review & Retrospective

## BIBLIOGRAFÍA

---



Para profundizar:

- <https://www.mountaingoatsoftware.com/blog>
- <https://medium.com/omarelgabrys-blog/requirements-engineering-requirements-validation-part-6-29778d7bde24>

- <https://openclassrooms.com/en/courses/4544611-write-agile-documentation-user-stories-and-acceptance-tests/4810081-writing-acceptance-tests>

## PDF DESCARGABLE

---

