POSIBLES PREGUNTAS DE FINAL

Introducción a las Metodologías del desarrollo del software

1. porque es necesario utilizar una metodología de desarrollo de software
2. Que es una metodologia de software
3. para qué sirve el analisis
4. Que es una metodologia de software
5. que es scrum
6. que es sprint
7. que es sprint retrospective
8. que se hace en la daily scrum
9. que es el product backlog
10. scrum master como es el rol como impacta en el proyecto
11. porque es necesario un scrum master en un equipo scrum
12. cual es el objetivo del sprint retrospective. Enumere los distintos detalles que aborda.
13. por que considera que realizar entregas del software funcionando cada cierto periodo corto es tan importante
14. Artefactos
    1. Cuáles son?
    2. Cómo se realizan/definen
    3. Quién/es es el responsable
15. Eventos:
    1. Cuáles son?
    2. Cuál es el objetivo de cada uno?
    3. Quiénes participan?
    4. Cuál es el tiempo que lleva cada uno?
16. scrum master como es el rol como impacta en el proyecto
17. porque es necesario un scrum master en un equipo scrum
18. cual es el objetivo del sprint retrospective. Enumere los distintos detalles que aborda.
19. explique el template de una user story indicando porque es util cada parte de la misma
20. por que considera que realizar entregas del software funcionando cada cierto periodo corto es tan importante
21. que es una user storie
22. ejemplo user storie como está formada
23. template de user storie
24. explique el template de una user story indicando porque es util cada parte de la misma
25. Las épicas se escribieron como Users Stories?
26. Cual es la granulalidad? Si es Épica, feature o User Story?
27. como esta formado el caso de uso
28. para qué sirve uml
29. que es UML
30. que diagramas usar en cada etapa
31. NO puede haber un actor sistema

**Final Introducción a las Metodologías de Desarrollo de Software**

11 de Agosto. Tiempo de examen 60 minutos.

1. Se está comenzando a desarrollar un proyecto que cuenta con un número elevado y complejo de casos de uso. ¿Qué circunstancias deberían darse para utilizar un ciclo de vida en cascada, en lugar de uno iterativo e incremental?

Debería ser un sistema para una maquina especifica que siempre funcione de la misma manera como un respirador o un llavero eléctrico. De todas maneras el ciclo en casca es completamente obsoleto ya que el iterativo e incremental hace más fácil el desarrollo del software y nos permite mostrar al cliente como se va desarrollando el producto y ver si este satisface sus necesidades.

2. ¿Por qué es necesario utilizar una metodología de desarrollo de software?

No solo que es necesario si no que si nos ponemos a analizar lo que ya hicimos seguramente seguimos algún tipo de metodología. La metodología es fundamental para la organización del trabajo y la distribución de las responsabilidades. Scrum, la metodología ágil utilizada en esta cátedra, nos provee un marco de trabajo fácil de aprender pero difícil de implementar a rajatabla.

3. Cuando quiero modelar la interacción entre objetos ¿qué diagrama UML debo utilizar? Describa en qué consiste y cómo se modela.

Utilizo el diagrama de secuencia ya que muestra la mecánica de la interacción con base en tiempos.

Rol de clases: Describe la manera en que un objeto se va a comportar.

Activaciones: Representan el tiempo que un objeto necesita para realizar una tarea determinada.

Mensajes: Que son flechas que representan comunicaciones entre objetos.

Linea de vida: Es la linea punteada que representa el tiempo de vida del objeto.

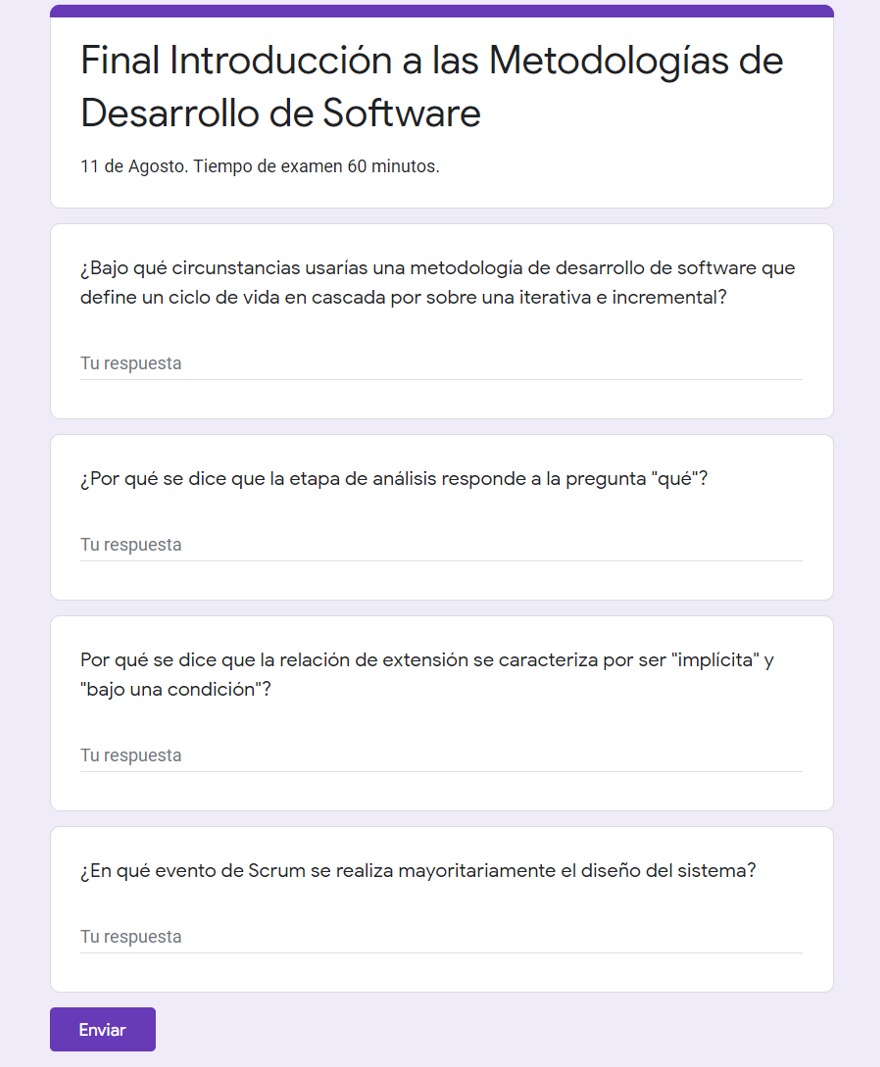
Destrucción: Un objeto puede ser destruido se envía el tag «Destruir»

Loop: Es un cuadro que nos permite representar un while, if, for o loop. Estos pueden ser ALT, OPT y LOOP.

4. ¿Qué significa que un evento sea Time-boxed en Scrum?

Los eventos de Scrum como Sprint Planning, Sprint, Daily Scrum están enmarcados por tiempo. El evento debe terminar tan pronto como se alcance el objetivo del evento o si el tiempo se termina.





**TP 1: METODOLOGIAS Y CICLO DE VIDA**

El software es intangible, la falta de principios y componentes bien definidos hace que un producto sea diferente de otro (si bien tienen la misma funcionalidad, 2 programas de stock hacen lo mismo pero no tienen que ser iguales) , es complejo por la existencia de cambios, evolución constante en su función o estructura, su desarrollo propenso a errores ,debido a esta complejidad es DESARROLLADO y no es construido.

El software es un conjunto de programas , procedimientos, reglas, documentación, datos.

Para organizarnos en el desarrollo del software utilizamos la Ingeniería del software, la Ing. Del Software me da herramientas para ordenarme, organizarme y me dice lo que tengo que hacer durante el desarrollo de software, me permite realizar un software de calidad.

El objetivo de la Ing del software es convertir el desarrollo del software en un proceso formal, con resultados predecibles, que permita obtener un producto final de calidad que satisfaga al cliente.

Hay varias definiciones diferentes de la Ingeniería del Software una de ellas la diferencia a través de disciplinas

**DISCIPLINA TECNICA**: Nos dice que es lo que tenemos que hacer para construir el producto o software

**DISCIPLINA DE GESTION**: Nos da guías para la construcción del producto para ayudarnos durante el proceso

**DISCIPLINA DE SOPORTE**: Nos da herramientas para construir un software de calidad

**DESARROLLO DE SOFTWARE ( 4 P )**

El software como producto se desarrolla en forma gradual y es el resultado de la ejecución de un proyecto.

El desarrollo de software se puede explicar con las 4 P: PERSONAS-PROYECTO -PROCESO -PRODUCTO

Las PERSONAS incorporan un PROYECTO el cual se adapta a un PROCESO ( el proyecto es la ejecución, desarrollo del del proceso ) ayudado por herramientas para obtener un PRODUCTO

**Personas** son los principales autores de un proyecto de Software, las personas asumen diferentes roles, conformando el Equipo de Desarrollo.

**Proyecto**: Es una ejecución única. Es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único, en este caso un producto de software.

Integra personas, utiliza proceso y herramientas para obtener como resultado un producto de software . Al principio se debe adaptar el proceso y el ciclo de vida al proyecto .

Un proyecto tiene las siguientes características:

* Temporal: Posee fecha de comienzo y finalización
* Productos, servicios o resultados únicos: Los resultados que se obtienen por similares que sean dos proyectos, tienen características que los hacen únicos
* Orientado a objetivos: Los objetivos deben ser claros, cuando todos comprenden lo que hay que lograr; y alcanzables cuando es factible de hacerse
* Elaboración gradual: Desarrollo en pasos que aumenta mediante incrementos

**Proceso** de software define un conjunto completo de actividades necesarias para transformar los requerimientos de un usuario en un producto

* Proporciona un marco de trabajo, una estructura que puede tomarse como referencia para definir el plan que guiará el proyecto
* Contiene además de la definición de las actividades, procedimientos y métodos, la identificación de las herramientas que permitan la automatización de las actividades y facilite el trabajo
* El proceso contiene actividades relacionadas con las disciplinas técnicas, de gestión y de soporte o protectoras
* El Ciclo de Vida del Proceso de Desarrollo de Software indica las fases/etapas del proceso y el orden en el que se llevarán a cabo

**Producto**: El resultado. Es el conjunto de artefactos o componentes que se obtienen como salida de cada una de las actividades definidas en un proceso, que se ejecutan en un proyecto

* Se obtiene como consecuencia de una evolución y refinamiento continuo de los modelos
* El producto es más que código, el producto que se obtiene es un sistema de software
* Un Artefacto/Diagrama/Modelo se refiere a cualquier documentación creada, producida, cambiada o utilizada por las personas en el desarrollo del sistema
* Un sistema de software es la sumatoria de todos los artefactos que se necesitan para representarlo en una forma comprensible para las máquinas, los trabajadores y los interesados

**Todo esto es ayudado por herramientas**

Ciclo de Vida

Definición

El modelo de Ciclo de Vida es una serie de pasos sistemáticos en etapas,

que facilita la comunicación y desempeño de los equipos de desarrollo de un sistema.

**Provee una representación estandarizada de las etapas del desarrollo de software, el**

**orden relativo entre estas etapas, los criterios de transición entre distintas etapas,**

**define el momento en que una etapa está completa y las precondiciones para iniciar**

**la etapa que sigue.**

Básicamente el modelo de ciclo de vida de desarrollo de software indica el orden de

realización de las distintas etapas y cuál será el producto obtenido en cada una a lo largo

del desarrollo del proyecto. Al momento de elegir el modelo influyen factores como el

tiempo, el tamaño del proyecto, la comunicación con el cliente, tamaño del equipo, entre

otros, de a cuerdo a las características del producto a desarrollar será el ciclo de vida que se debe utilizar.

**ETAPAS DEL CICLO DE VIDA**

1. Captura de requerimentos: Es obtener que es lo que quiere el cliente y que necesidades tiene para desarrollar el sistema, es abstraer la información necesaria para desarrollar el sistema y a su vez delimitar el sistema, que es lo que se va a desarrollar y que cosas se van a dejar afuera( por ejemplo viene un cliente que tiene un presupuesto acotado, entonces se desarrolla lo que da ese presupuesto ). Se entiende el dominio en el que se va a trabajar y se ve cuál es la funcionalidad del sistema. **DIAGRAMAS CASOS DE USO, DIAGRAMA DE ACTIVIDADES**

HAY REQUERIMENTOS FUNCIONALES Y NO FUNCIONALES

Los requerimentos funcionales son como las opciones del menú es algo que se empieza y termina en un periodo de tiempo corto. Estos son los que se toman en la captura de requerimentos( hacen a la ejecución del sistema )

Los requerimentos no funcionales son restricciones que atraviesan transversalmente a todos o algunos requerimentos funcionales ( ejm si uso BD relacional o BD orientada a objetos si el sistema es centralizado o distribuido, etc ). Los requerimentos no funcionales no se los captura, se los deja anotado y se trabajan en la etapa de diseño

POR ESTA RAZON LA CAPTURA DE REQUERIMENTOS Y EL ANALISIS DICEN QUE ES LO QUE TIENE QUE HACER EL SISTEMA ( PLASMO O REALIZO LA CAPTURA DE REQUERIMENTOS)

Y LA ETAPA DE DISEÑO ME DICE COMO VOY A RESOLVER EL SISTEMA, COMO RESULEVO LOS REQ FUNCIONALES Y COMO RESULEVO LOS REQ NO FUNCIONALES

TECNICAS DE ESPECIFICACION DE REQUERIMENTOS

CASOS DE USO: Nos permite especificar requerimentos y delimitar el sistema

HISTORIAS DE USUARIO: Nos permite ver de una forma mas informal que Casos de Uso cual es la funcionalidad que debe tener el sistema.

1. Análisis y Diseño: Se modelan los requerimentos funcionales o requerimentos del cliente( análisis, acá se dice **qué** es lo que tiene que hacer el sistema , **DE ACTIVIDADES, DE COLABORACION**) y luego se modela la solución del sistema, teniendo en cuenta el ambiente de implementación, esto es: Si es un sistema centralizado o distribuido, la base de datos, el lenguaje de programación o el nivel de performance deseado, etc. ( diseño, acá se dice **cómo** voy a resolver el sistema ). **DIAGRAMAS CLASES Y DIAGRAMA DE SECUENCIA, COMPONENTES, DEPLOYMENT, TRANSICION DE ESTADOS**
2. Implementación: Se implementa la solución con el/los lenguajes de programación adecuados. Involucran todos los diagramas UML usados en diseño pero de manera visual, para mantener el enfoque y consistencia del sistema.
3. Testeo: Se realizan pruebas para verificar y validar el sistema según los criterios correspondientes. **DIAGRAMAS CLASES Y DIAGRAMA DE SECUENCIA**
4. Mantenimiento: Es la etapa más difícil del desarrollo del sistema, en esta se actualiza o modifica el sistema si surgen nuevos requerimientos.

**Modelos de Ciclos de vida de un Sistema**

Para el diseño y desarrollo de un producto de software se aplican metodologías, modelos y técnicas que permiten resolver problemas .

Se han definido varias metodologías de desarrollo que ayudan a desarrollar software de calidad .

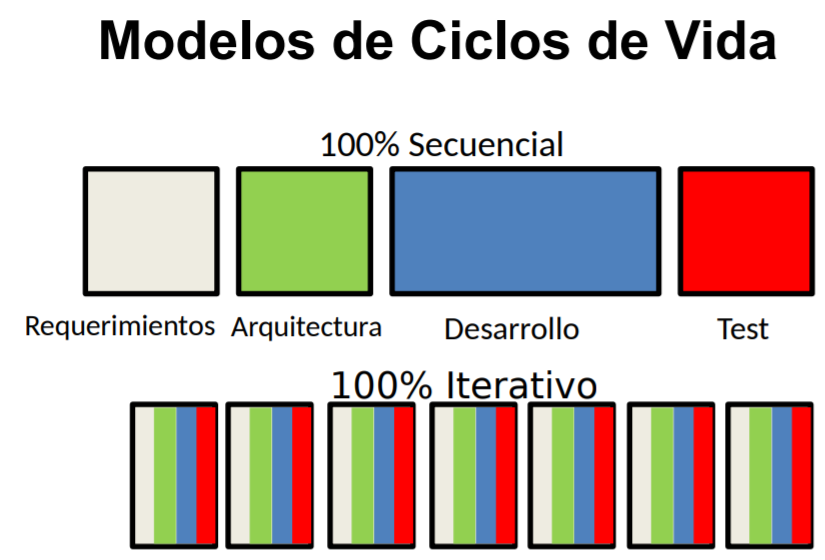
Cada modelo representa un proceso desde una perspectiva particular y así proporciona información parcial sobre el proceso

Cada modelo de ciclo de vida ayuda al desarrollador a ordenar el trabajo a ser realizado durante la construcción del producto

Cada modelo de ciclo de vida indica en qué orden se desarrollarán las etapas del ciclo de vida

Existen varios tipos de Ciclos de Vida :

* Secuencial ( en un principio se usaba este)
* Iterativo/Incremental
* etc.



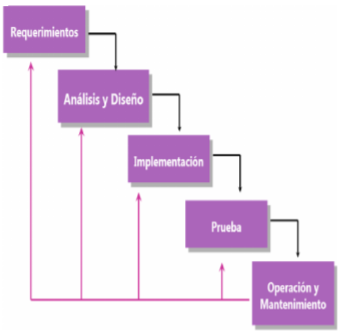
**Modelo de Ciclo de Vida en Cascada**

Considera que las actividades (o etapas) del proceso de desarrollo: Requerimientos, análisis, diseño, implementación y prueba, son etapas separadas que para continuar con la siguiente se debe completar totalmente la anterior

La retroalimentación con el cliente se realiza una vez que el producto ha sido terminado

Es muy costoso volver a las etapas anteriores para realizar modificaciones por un malentendimiento de los requerimientos.

Sirve cuando los requerimentos son fijos , muy estables, muy detallados y se tiene bien en claro lo que se tiene que hacer.



**Modelo de Ciclo de Vida Incremental**

El software se desarrolla gradualmente, por funcionalidades que incrementan el producto

Se divide el problema en varios subproblemas y se desarrolla por completo ( un subproblema o subsistema) hasta llegar a la funcionalidad de dicho subsistema y se junta con la funcionalidad del subsistema creado anteriormente, para obtener un incremento del producto final.

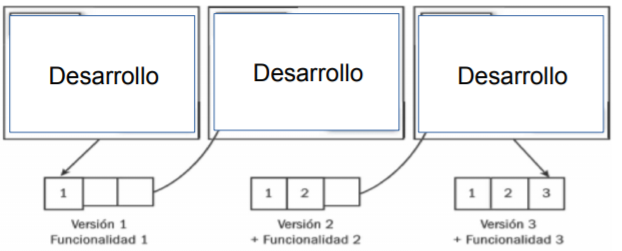
Luego de cada ciclo es posible realizar una entrega de software parcial al cliente

• En este modelo se planifica el sistema en bloques de funcionalidad (incrementos).

• Cada incremento es analizado según el ciclo de vida en cascada

• Cada incremento añade funcionalidad al sistema, pero no tiene porqué ser

usable en cada uno.



**Modelo de Ciclo de Vida Iterativo**

El problema es dividido en subproblemas más pequeños

Una iteración es la aplicación de un ciclo de vida en cascada a un subproblema identificado

El objetivo es tener en claro la problemática que se está intentando resolver

• Se planifica el proyecto en varios bloques temporales, llamados iteraciones, y a

cada uno se lo analiza desde el ciclo de vida en cascada.

• Es recomendable que las iteraciones duren de 2 a 6 semanas, ya que es mejor

tener objetivos a corto plazo para el equipo de desarrollo, además de que se

tiene mejor precisión en las estimaciones.

• También es recomendable definir iteraciones regulares, así actúan como

timebox de la misma duración.

• Los timebox aportan beneficios al equipo en general:

o El equipo de desarrollo aprende a calcular su tiempo de trabajo sin

necesitar extrapolar de otras iteraciones

o El cliente puede proyectar cuantas iteraciones se necesitan para cada

entrega y tomar decisiones al respecto

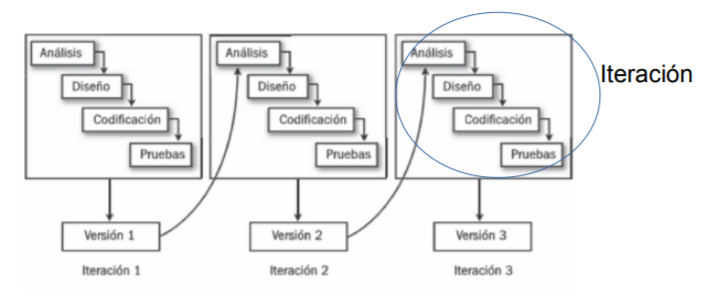
o Permiten gestionar y sincronizar de manera sencilla las necesidades del

proyecto con respecto a la de otros proyectos.

• Se obtienen prototipos usables e independientes desde el primer ciclo iterativo,

en la segunda versión de la siguiente iteración el software seguirá siendo usable

y además estará mejorado.



**Modelo de Ciclo de Vida Iterativo Incremental**

• Es una conjunción de los dos modelos anteriores.

• Cada iteración se puede entender como un mini proyecto, que entrega un

incremento sobre el producto final, en cada iteración se aplican pequeños ciclos en cascada.

• El incremento se debe completar al final de cada iteración, el equipo debe

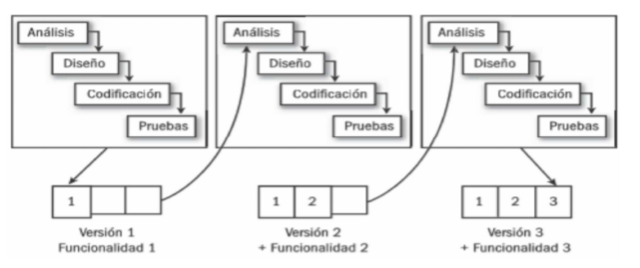
realizar todas las tareas necesarias para completarlo y debe poder ser

entregado al cliente.

• La elección de lo que se implementara en cada iteración se basa en el conjunto

de requerimientos que amplían la funcionalidad y en los riesgos más

importantes que se deben debilitar.



**METODOLOGIA DEL DESARROLLO DEL SOFTWARE**

**Metodología de Desarrollo de Software Definición Metodología de desarrollo de software es un proceso organizado para la producción de software, define una forma disciplinada para desarrollar software con el objetivo de hacerlo más predecible y eficiente, donde se especifica qué ciclo de vida se va a utilizar, qué personas desempeñaran cada rol en el desarrollo de las actividades cómo las etapas del ciclo de vida que se desarrollarán, y los artefactos a generar durante el desarrollo del producto,. Es necesario utilizar una metodología cuando el proyecto es grande y complejo, pero además cuando hay más de tres personas involucradas se logra una buena comunicación y buen desempeño.**

**Hay formas diferentes formas de organizarnos, la metodología nos dice como tenemos que organizarnos, nos dice cual es el ciclo de vida que hay que utilizar para desarrollar un sistema, y a su vez me dice cuáles son los artefactos, los documentos que hay que generar en cada una de las etapas del desarrollo del software.**

Proceso de Desarrollo de Software: Conjunto de actividades necesarias para llevar a cabo el proyecto.

Rol: Papel que desempeña una persona en el desarrollo del software, más adelante se detallan los roles que puede haber.

De acuerdo a las características del producto se definirá la metodología de desarrollo de software a utilizar. Existen varios tipos de metodologías:

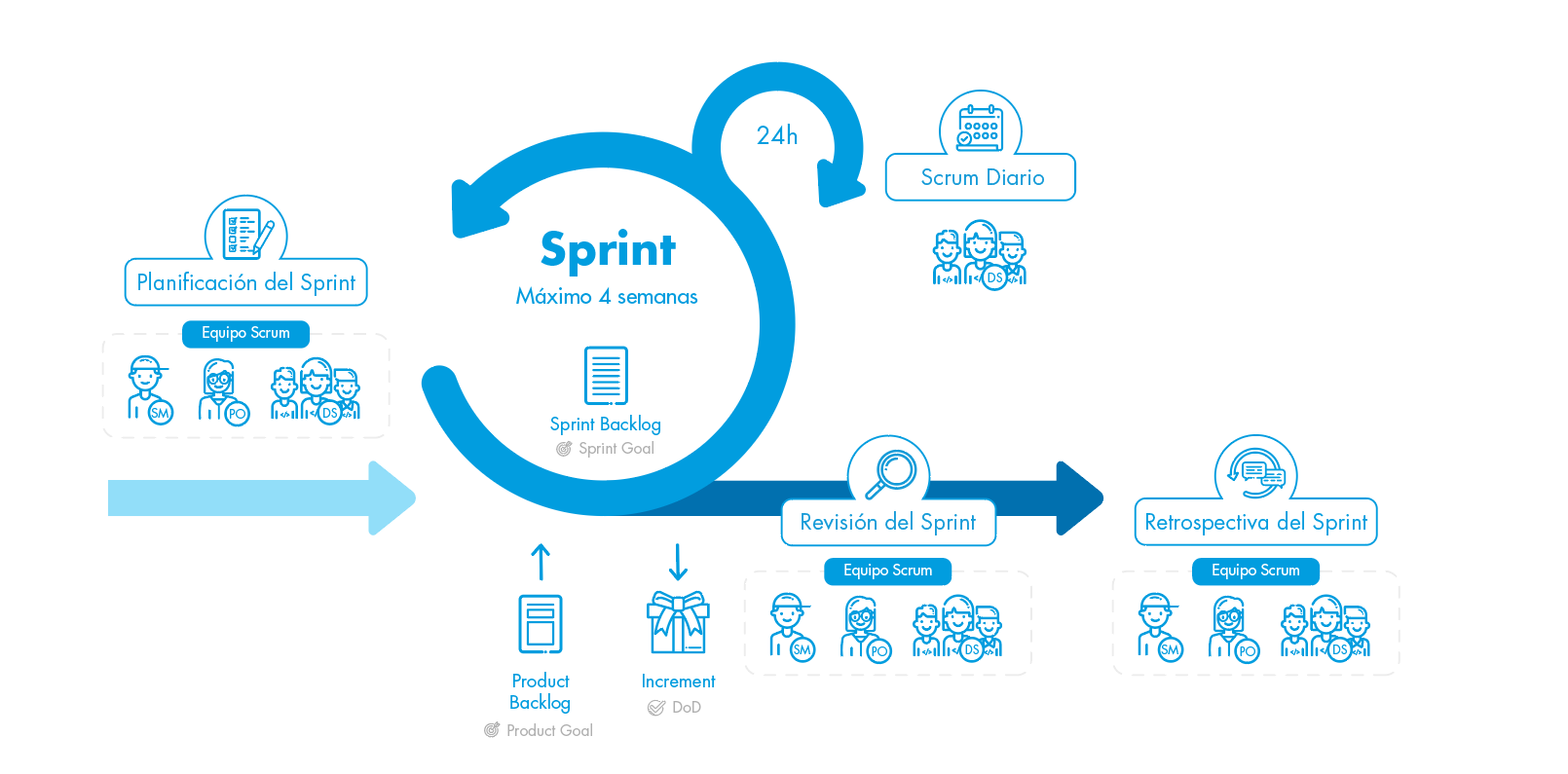
Tipos de Metodologías de Desarrollo de Software

* Metodologías Estructuradas Las s funciones y los datos ( acá estaban definidas las funciones por un lado y los datos por otro lado, es el paradigma procedural) ,se tratan como entidades independientes, estos sistemas resultan difíciles de mantener, las modificaciones en la estructura de los datos conllevan mucho trabajo y además una persona no piensa generalmente en términos de estructuras, por lo que los requisitos se toman en lenguaje natural, especificando más bien funcionalidad y se dejan las estructuras para etapas de análisis y diseño. Orientado a función/dato, ASML. Sistemas de información clásicos
* Metodologías Orientadas a Objetos los datos y las funciones junto con funciones están encapsulados en entidades fuertemente relacionadas,que son los objetos. Provee un fácil mantenimiento y extensión. Orientado a Objetos, UML, UP, Ciclo de Vida iterativo e incremental.
* Metodologías Formales Surgen como enfoques analíticos en donde el desarrollo del software puede ser verificado usando demostraciones matemáticas, es recomendable cuando el sistema a desarrollar sea de alto riesgo, es necesario que todos los componentes (hardware, software, usuarios) se comporten de la manera correcta ante situaciones específicas. B, Z, Aplicaciones Militares, Medicina, Planificación de Trafico.
* Metodologías Agiles Surge debido a que los sistemas empiezan a ser cambiantes, y las metodologías orientadas a objeto ya no alcanzaban o no servían porque son más rígidas. Dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Scrum, Sprint

**Proceso de Desarrollo de Software**

* **Una proceso de desarollo de software define la metodologia de desarrollo de software a utilizar.**
* **Una metodología de desarrollo de software define el ciclo de vida que se utilizará , los artefactos que se deben generar en cada una de las etapas del ciclo de vida y los roles necesarios para el desarrollo.**
* **El ciclo de vida de un sistema define el orden en el cuál las etapas del ciclo de vida se desarrollarán**
* **Las etapas del ciclo de vida definen las actividades necesarias para desarrollar un sistema**

**SCRUM**



**Definición**

1. **Scrum es un marco de trabajo para desarrollar, entregar y mantener productos complejos, por el cual las personas pueden abordar problemas complejos adaptativos, a la vez que entregar productos del máximo valor posible productiva y creativamente.**
2. **Consiste en los equipos de Scrum y sus eventos, roles, artefactos, y reglas asociadas entre ellos que rigen la interacción entre ellos.**
3. **Se basa en PLANIFICAR (lo que se va a hacer)-EJECUTAR (lo planificado)-REFLEXIONAR (sobre cómo se realizó esa ejecución y si se puede mejorar tanto la ejecución como la planificación) en un tiempo corto de vida (30 días o menos).**
4. **Scrum no es una metodología es un camino, pero se la considera una metodología porque nos da un ciclo de vida, nos da los artefactos que hay que generar a lo largo del ciclo de vida, también nos da un conjunto de roles que se necesitan para desarrollar un sistema.**
5. **Scrum es sencillo y se puede entender e implementar en unos pocos días**

**Características**

1. Equipo autodirigido y autoorganizado (el DT tiene el poder de autoorganizarse cada miembro puede elegir en que va a trabajar dentro de una iteración).
2. No hay adición externa de trabajo a una iteración, una vez elegida (no se le puede agregar más trabajo porque esta se planifica).
3. Tiene una Reunión diaria con preguntas especiales.
4. Demostración a las partes interesadas externas al final de cada iteración.
5. Cada iteración está guiada o conducida por el cliente es decir que el cliente es el que decide que es lo que hay que hacer, todo consensuado con el resto del equipo.

**Pilares de Scrum**

- **Transparencia**

- los aspectos del proceso deben ser visibles y claros para el cliente, el cliente debe estar todo el tiempo consiente de cuál es el estado del proyecto.

Los aspectos significativos del proceso deben ser visibles para aquellos que son responsables del resultado. La transparencia requiere que dichos aspectos sean definidos por un estándar común, de tal modo que los observadores compartan un entendimiento común de lo que se están viendo. Por ejemplo:

• Todos los participantes deben compartir un lenguaje común para referirse al proceso; y,

• Aquellos que desempeñan el trabajo y quienes inspeccionan el incremento resultante deben compartir una definición común de “Terminado”.

La transparencia se mide con la transparencia de los artefactos, no tiene que haber diferencias entre los resultados esperados y los reales.

- **Inspección** (de los artefactos y del proceso o metodología)

- Debe realizarse con frecuencia para detectar desviaciones no deseadas pero a la vez no tan frecuentes como para que interfieran en el trabajo, se realiza para ver cómo vamos, como nos estamos comunicando, como trabajamos.

Se realiza la inspección para luego hacer una adaptación.

- **Adaptación**

- Cuando el proceso debe ser ajustado porque hay una desviación y el producto resultante va a ser inaceptable. Este ajuste debe hacerse cuanto antes para minimizar desviaciones mayores.

- Scrum presenta 4 eventos para la inspección y adaptación:

1. Planificación del Sprint( Sprint planning )
2. Scrum diario (Daily Scrum)
3. Revisión del Sprint (Sprint Review)
4. Restrospectiva del Sprint (Sprint Retrospective)

**Valores de Scrum**

Scrum está centrado en el equipo de desarrollo, si hay algo por lo que falla SCRUM es por la falta de compromiso del equipo, entonces se espera que el equipo comparta ciertos valores.

1. Coraje: Es hacer lo correcto aunque sea difícil
2. Enfoque: Esta centrado en el objetivo de la iteración y cumplirla
3. Compromiso: Todo los miembros deben estar comprometidos en hacer el trabajo necesario
4. Respeto: Entre los distintos miembros del equipo
5. Apertura: Es estar dispuestos a cambios que puedan surgir y adaptarnos a esos cambios.

Cuando el Equipo Scrum incorpora y vivencia los valores de compromiso, coraje, foco, apertura y respeto, los pilares Scrum de transparencia, inspección y adaptación se materializan y fomentan la confianza. Los miembros del Equipo Scrum aprenden y exploran estos valores a medida que trabajan en los eventos, roles y artefactos de Scrum.

El uso exitoso de Scrum depende de que las personas lleguen a ser más virtuosas en la convivencia con estos cinco valores. Las personas se comprometen de manera individual a alcanzar las metas del Equipo Scrum. Los miembros del Equipo Scrum tienen coraje para hacer bien las cosas y para trabajar en los problemas difíciles. Todos se enfocan en el trabajo del Sprint y en las metas del Equipo Scrum. El Equipo Scrum y sus interesados acuerdan estar abiertos a todo el trabajo y a los desafíos que se les presenten al realizar su trabajo. Los miembros del Equipo Scrum se respetan entre sí para ser personas capaces e independientes.

**Cuando no se usa SCRUM**:

1. Cuando hay un PO que no está comprometido con el desarrollo, por ejm pide que le entreguen un trabajo llave en mano.
2. Cuando No hay compromiso en el DT.
3. Cuando se está en mantenimiento, cuando ya hay un sistema que está en producción , más que nada debido que en estos casos requieren agregar o modificar tareas sobre la marcha y SCRUM no contempla esto, ya que una vez comenzado el sprint no se le puede agregar más trabajo. (para esto se puede utilizar 2 técnicas que son : Lean o Kanban)

**ROLES DE SCRUM ( SCRUM TEAM )**

**Product Owner :**

1. Es el representante del cliente, es el que sabe el sistema en el que se va a trabajar.
2. Es el responsable del proyecto del lado del cliente
3. debe ser una única persona (para que no haya malos entendidos con terceros).
4. debe estar disponible para que pueda hablar con el DT.
5. Es el responsable del gerenciamiento del Product Backlog,
6. debe expresar, priorizar y ayudar a entender los requerimentos.
7. Es el que establece el objetivo de cada iteración.

Para que el Dueño de Producto pueda hacer bien su trabajo, toda la organización debe respetar sus decisiones. Las decisiones del Dueño de Producto se reflejan en el contenido y en la priorización de la Lista del Producto

**Development Team:**

1. Es el que va a hacer el trabajo de entregar un incremento de producto terminado
2. es un grupo de 5 a 9 personas) Este número es el que está estipulado para que el equipo funcione bien.
3. Se autogobierna,
4. no hay etiquetas todos tiene que estar disponibles para codear, y todos deben saber hacer todo.
5. No hay subequipos que se encarguen algo en particular, todo lo hace el DT.
6. Los equipos son multifuncionales.
7. Los Miembros individuales del Equipo de Desarrollo pueden tener habilidades especializadas y áreas en las que estén más enfocados, pero la responsabilidad recae en el Equipo de Desarrollo como un todo.

**Scrum Master:**

El Scrum Master es responsable de promover y apoyar Scrum como se define en la Guía de Scrum. Los Scrum Masters hacen esto ayudando a todos a entender la teoría, prácticas, reglas y valores de Scrum.

El Scrum Master es un líder que está al servicio del Equipo Scrum. El Scrum Master ayuda a las personas externas al Equipo Scrum a entender qué interacciones con el Equipo Scrum pueden ser útiles y cuáles no.

El Scrum Master ayuda a todos a modificar estas interacciones para maximizar el valor creado por el Equipo Scrum.

1. Es el facilitador,
2. es el que sabe bien la metodología y va a ayudar a cumplir con la práctica de scrum, los valores de scrum, las reglas de scrum,
3. se encarga de que todos asistan a los eventos que les corresponden,
4. se encarga de la entrega de los artefactos en cada etapa
5. va a guiar en que se realice bien los eventos de la metodología.
6. Un Scrum Master puede tener asignado más de un proyecto.

**El Servicio del Scrum Master al Dueño de Producto**

El Scrum Master da servicio al Dueño de Producto de varias formas, incluyendo:

• Asegurar que los objetivos, el alcance y el dominio del producto sean entendidos por todos en el equipo Scrum de la mejor manera posible;

• Encontrar técnicas para gestionar la Lista de Producto de manera efectiva;

• Ayudar al Equipo Scrum a entender la necesidad de contar con elementos de Lista de Producto claros y concisos;

• Entender la planificación del producto en un entorno empírico;

• Asegurar que el Dueño de Producto conozca cómo ordenar la Lista de Producto para maximizar el valor;

• Entender y practicar la agilidad; y,

• Facilitar los eventos de Scrum según se requiera o necesite.

**El Servicio del Scrum Master al Equipo de Desarrollo**

El Scrum Master da servicio al Equipo de Desarrollo de varias formas, incluyendo:

• Guiar al Equipo de Desarrollo en ser autoorganizado y multifuncional;

• Ayudar al Equipo de Desarrollo a crear productos de alto valor;

• Eliminar impedimentos para el progreso del Equipo de Desarrollo;

• Facilitar los eventos de Scrum según se requiera o necesite; y,

• Guiar al Equipo de Desarrollo en entornos organizacionales en los que Scrum aún no haya sido adoptado y entendido por completo.

**El Servicio del Scrum Master a la Organización**

El Scrum Master da servicio a la organización de varias formas, incluyendo:

• Liderar y guiar a la organización en la adopción de Scrum;

• Planificar las implementaciones de Scrum en la organización;

• Ayudar a los empleados e interesados a entender y llevar a cabo Scrum y el desarrollo empírico de producto;

• Motivar cambios que incrementen la productividad del Equipo Scrum; y,

• Trabajar con otros Scrum Masters para incrementar la efectividad de la aplicación de Scrum en la organización.

**Eventos de Scrum**

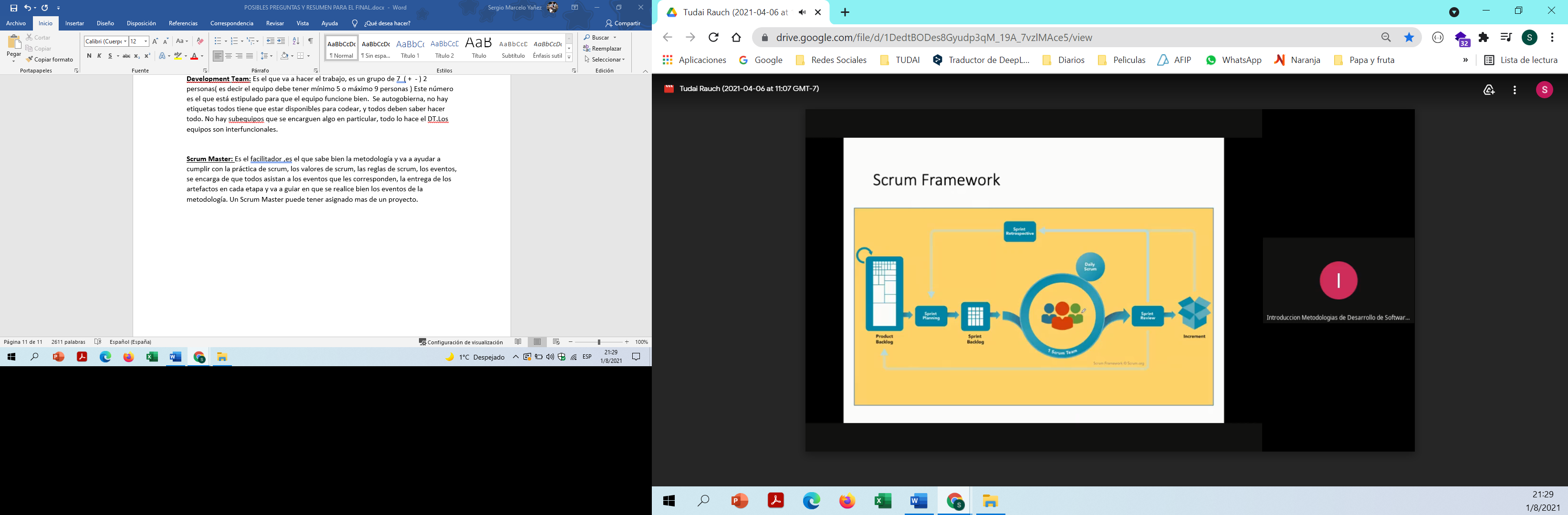
En Scrum existen eventos predefinidos con el fin de crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Todos los eventos son bloques de tiempo (time-boxes), de tal modo que todos tienen una duración máxima. Una vez que comienza un Sprint, su duración es fija y no puede acortarse o alargarse. Los demás eventos pueden terminar siempre que se alcance el objetivo del evento, asegurando que se emplee una cantidad apropiada de tiempo sin permitir desperdicio en el proceso.

Cada uno de los eventos de Scrum constituye una oportunidad formal para la inspección y adaptación de algún aspecto. Estos eventos se diseñaron específicamente para habilitar los pilares vitales de transparencia e inspección. La falta de alguno de estos eventos da como resultado una reducción de la transparencia y constituye una oportunidad perdida de inspección y adaptación.

**Artefactos de Scrum**

Los artefactos de Scrum representan trabajo o valor en diversas formas que son útiles para proporcionar transparencia y oportunidades para la inspección y adaptación. Los artefactos definidos por Scrum están diseñados específicamente para maximizar la transparencia de la información clave, necesaria para asegurar que todos tengan el mismo entendimiento del artefacto.

**Ciclo de vida o Framework de SCRUM**



Sprint

Artefacto

Artefacto

Artefacto

Evento

Evento

Evento

Eventoto

Evento

El ciclo de vida o framework de scrum tiene Artefactos y eventos

**Product Backlog: es un artefacto ,** es la lista de todos los requerimentos que hay que implementar en el sistema. Es la lista ordenada, priorizada por el PO ( lista de PBIs ).

El producto backlog se construye en el Warm-up que es un pre-sprint anterior al primer sprint, es corto de 1 semana 10 días no más de 2 semanas además se hace un prototipo que `puede ser descartable o no que luego puede ir evolucionado. Acá los requerimentos se pasan a lenguaje de Users Stories. Participan el PO, SM, DT + stakeholders.

**Lista de Producto (Product Backlog)**

La Lista de Producto es una lista ordenada de todo lo que se conoce que es necesario en el producto. Es la única fuente de requisitos para cualquier cambio a realizarse en el producto. El (Product Owner) es el responsable del Product Backlog, incluyendo su contenido, disponibilidad y ordenación.

Una Lista del Product Backlog nunca está completa. El desarrollo más temprano de la misma solo refleja los requisitos conocidos y mejor entendidos al principio. La Lista del Product Backlog evoluciona a medida de qué el producto y el entorno en el que se usará también lo hacen. La Lista del Product Backlog es dinámica; cambia constantemente para identificar lo que el producto necesita para ser adecuado, competitivo y útil. Si un producto existe, su Product Backlog también existe.

El Product Backlog enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que constituyen cambios a realizarse sobre el producto para entregas futuras. Los elementos del Product Backlog tienen como atributos la descripción, el orden, la estimación y el valor. Los elementos de Product Backlog muchas veces incluyen descripciones de las pruebas que demostrarán la completitud de tales elementos cuando estén “Terminados”.

A medida que un producto es utilizado y se incrementa su valor y el mercado proporciona retroalimentación, el Product Backlog se convierte en una lista más larga y exhaustiva. Los requisitos nunca dejan de cambiar así que la El Product Backlog es un artefacto vivo. Los cambios en los requisitos de negocio, las condiciones del mercado o la tecnología podrían causar cambios en la Lista de Producto.

Los elementos de la Product Backlog de orden más alto son generalmente más claros y detallados que los de menor orden. Se realizan estimaciones más precisas basándose en la mayor claridad y detalle; cuanto más bajo es el orden, menor es el detalle. Los elementos de la Product Backlog de los que se ocupará el Equipo de Desarrollo en el siguiente Sprint tienen una granularidad mayor, habiendo sido descompuestos de forma que cualquier elemento pueda ser “Terminado” dentro de los límites del bloque de tiempo del Sprint. Los elementos del Product Backlog que pueden ser “Terminados” por el Equipo de Desarrollo en un Sprint son considerados “Preparados” o “accionables” para ser seleccionados en una reunión de Planificación de Sprint. Los elementos de la Product Backlog normalmente adquieren este grado de transparencia mediante las actividades de refinamiento

Sprint Planning: es un evento,

. **Es time-box, max 8hs**. para un sprint de un mes, **participa todo el equipo scrum(PO-SM-DT ).**

La Planificación de Sprint responde a las siguientes preguntas:

• ¿Qué puede entregarse en el Incremento resultante del Sprint que comienza?

• ¿Cómo se conseguirá hacer el trabajo necesario para entregar el Incremento?

**Durante la Planificación del Sprint, el Equipo Scrum también define un Objetivo del Sprint (Sprint Goal**). El Objetivo del Sprint debería lograrse durante el Sprint a través de la implementación de la Lista de Producto y proporciona una guía al equipo de desarrollo de por qué se está construyendo el incremento.

Es una reunión donde el PO decide la prioridad de los requerimentos y se decide la cantidad de requerimentos que se pueden hacer en el próximo Sprint ya que depende de la capacidad que tenga el DT, acá se calcula la capacidad y la velocidad del DT.

Se le realizan todas las preguntas necesarias al PO para obtener los detalles. De ésta reunión se obtiene el Sprint backlog. El DT analiza como lo va a hacer y estima cada PBIs que se eligió para el sprint backlog

Acá se realizan las **estimaciones** de las User Stories, se determina cuánto tiempo va a llevar implementar la User Storie. El que estima es el DT, el PO participa de la reunión pero no estima, el SM hace de moderador pero no estima. Se estima en horas o en Story points. La técnica más usada para estimar se llama Planning Poker que se estima con cartas, cada desarrollador tiene un mazo de cartas.

El moderador lee la User Story, el PO responde las preguntas que tenga el equipo, luego cada desarrollador elige secretamente una carta que representa su estimación, luego de que todos seleccionaron se muestran todas las cartas. Si las estimaciones son muy diferentes se les pide que explique su razonamiento de por que estimaron tan alto o bajo, la votación se repite hasta tener consenso.

Suele usarse un pivote para referencia, por ejm se utiliza como pivote la user más fácil o que tenga menos puntos

**Objetivos del sprint planning:**

* **Tener PBIs estimados de alta precisión**
* **Tener un mejor entendimiento del trabajo y dividir los PBIs para generar ítems más pequeños**
* **Obtener el Sprint Backlog**

Sprint backlog: Es un artefacto, es el conjunto de requerimentos que se van a implementar en el próximo sprint. Este conjunto responden a un objetivo del sprint ( GOAL del sprint)

La Lista de Pendientes del Sprint hace visible todo el trabajo que el Equipo de Desarrollo identifica como necesario para alcanzar el Objetivo del Sprint

Cuando se requiere nuevo trabajo, el Equipo de Desarrollo lo adiciona a la Lista de Pendientes del Sprint. A medida que el trabajo se ejecuta o se completa se va actualizando la estimación de trabajo restante. Cuando algún elemento del plan se considera innecesario, es eliminado. Solo el Equipo de Desarrollo puede cambiar su Lista de Pendientes del Sprint durante un Sprint. La Lista de Pendientes del Sprint es una imagen visible en tiempo real del trabajo que el Equipo de Desarrollo planea llevar a cabo durante el Sprint y pertenece únicamente al Equipo de Desarrollo.

En esta parte el PO no puede modificar el sprint backlog, pero el equipo de desarrollo si puede hacerlo para aclarar algún detalle. Para organizarlo se utiliza un SCRUM-BOARD, en nuestro caso usamos JIRA. Esto es lo que permite que scrum sea autodirigido y autoasignado. Cada miembro elige una tarjeta con una tarea y la realiza.

Sprint( es una iteración) : es un evento, es el corazón de Scrum el cual se crea un incremento de producto terminado, utilizable y potencialmente desplegable. Acá se toman los requerimentos del sprint backlog, se realiza el análisis y diseño, se testea. Todo aplicando cascada. **Es time-box, lleva un tiempo de 3 a 4 semanas**, raramente se cancela debido a que son cortos, si se cancela sólo el PO puede hacerlo, aunque puede hacerlo bajo influencia de los interesados. Un Sprint se cancelaría si el Objetivo del Sprint llega a quedar obsoleto. Tiene un tiempo fijo, se determina que día va a terminar. Todos los días se realiza una reunión ( Daily Scrum) para saber cómo se va y en que está trabajando cada miembro del equipo. El objetivo del sprint es implementar el sprint backlog obteniendo un incremento.

Los Sprints contienen y consisten en la Planificación del Sprint (Sprint Planning), los Scrums Diarios (Daily Scrums), el trabajo de desarrollo, la Revisión del Sprint (Sprint Review), y la Retrospectiva del Sprint (Sprint Retrospective).

. Cada nuevo Sprint comienza inmediatamente después de la finalización del Sprint anterior.

Daily Scrum o Daily Meeting: es un evento ,

**Es time-box (15 min**). **Solo participan los miembros del DT,** , **el Equipo de Desarrollo planea el trabajo para las siguientes 24 horas**

Se realiza todos los días del sprint para saber cómo les va y en que está trabajando cada miembro del equipo. Cada miembro responde que hizo, que va a hacer, si esta trabado con algo.**.** El SM no participa de la reunión pero se encarga de que todo el DT lo haga. Nadie debe llegar tarde, impone sanciones. Por ejm se define que sea todos los días a las 9 de la mañana **para evitar complejidad de organizarla y generar una costumbre.**

Esta reunión permite optimizar las posibilidades de que el DT cumpla con el objetivo del sprint, mejoran la comunicación, eliminan la necesidad de realizar otras reuniones, identifican impedimentos a remover relativos al desarrollo, resaltan y promueven la toma rápida de decisiones y mejoran el nivel de conocimiento del Equipo de Desarrollo. El Scrum Diario es una reunión clave de inspección y adaptación.

Sprint Review: es un evento,

**Participa todo el equipo scrum ( PO, SM, DT )+ stakeholders . Es time-box, dura 4 hs. para un sprint de 1 mes.**

Se realiza para inspeccionar el Incremento y adaptar la Lista de Producto si fuese necesario. Está enfocada en el producto.

Se hace una presentación del estado del proyecto según el PO, se hace una demo del sistema con el cliente que tiene como objetivo facilitar la retroalimentación de información y fomentar la colaboración., el DT recibe feedback ( el DT ve como le está yendo, si tiene que cambiar algo o si va bien así).

Es una reunión con mucho protocolo.

El Product Owner explica qué elementos del Product Backlog han sido"Hecho" y lo que no ha sido "Hecho".

- El Equipo de Desarrollo habla de lo que ha ido bien durante el Sprint, qué problemas se encontraron y cómo se resolvieron.

- El Equipo de Desarrollo demuestra el trabajo que ha "Hecho" y responde a las preguntas sobre el incremento. El DT anota las objeciones y las envía al Product backlog para un próximo sprint

- El PO discute el Product Backlog para el próximo sprint aprovechando que están los Stakeholders

- El grupo colabora en lo que debe hacer a continuación (planificación del Sprint de entrada)

- Revisión de los posibles cambios en el mercado, el calendario, el presupuesto, etc.

El resultado de la Sprint Review es una Lista de Producto revisada que define los elementos de la Lista del Product Backlog, posibles para el siguiente Sprint. Es posible además que la Lista de Product Backlog reciba un ajuste general para enfocarse en nuevas oportunidades.

Incremento: es un artefacto, es el software que se obtuvo del sprint, junto con toda su documentación. Es la suma de todos los PBIs que se completaron en el último sprint sumados a los sprints anteriores, el incremento debe estar en DONE, y debe ser usable, el cliente debería poder ponerlo en producción si quiere.

. Un incremento es un cuerpo de trabajo inspeccionable y terminado que respalda el lo hecho al final del Sprint.

Sprint Retrospective: es un evento,

**Participan el SCRUM team ( PO, SM, DT ), Es time-box, 3 hs**. para un sprint de 1 mes.

Acá se reflexiona a partir de lo que se hizo en el último sprint, que se hizo bien, que se hizo mal, que se puede mejorar.

Está enfocada en el DT y cómo realizó esta parte del proceso, se evalúa como trabajo el DT , cómo se interactuó con el PO, que mejoras pueden hacerse.

Una vez terminado con esto volvemos al sprint planning para armar el próximo sprint.

El **objetivo de la sprint retrospective es:** la mejora continua del proceso seguido por los miembros del equipo.

Para el final de la Sprint Retrospective el Equipo Scrum debería haber identificado mejoras que implementará en el próximo Sprint. El hecho de implementar estas mejoras en el siguiente Sprint constituye la adaptación subsecuente a la inspección del Equipo de Desarrollo mismo. Aunque las mejoras pueden implementarse en cualquier momento, la Retrospectiva de Sprint ofrece un evento dedicado para este fin, enfocado en la inspección y la adaptación.

**CRITERIO DE DONE**

Para saber cuándo una tarea està terminada o en Done, se acuerda con el cliente el significado de terminado ( DoD Definición de Done), en base a este acuerdo se realiza el Criterio de Done que va a regir durante todo el desarrollo del sistema, va a ser único por proyecto y debe respetarse a rajatabla en todos sus aspectos.

Que sea Done significa que està listo para ejecutar

* DoD tiene 3 componentes principales:
* Los requerimentos funcionales( deben estar cumplidos ).
* Cuestiones de calidad ( es decir que se cumpla con: standard de código, condiciones de testing, índices de mantenibilidad, etc ).
* Los requerimentos no funcionales también deben cumplirse.
* El DoD es a nivel de producto ( significa que el producto esté terminado ).
* Cada requerimento o tarea debe cumplir con la definición de Done.

**PBIs EN SCRUM**

El Product Backlog es la lista ordenada de todos los requerimentos del sistema, hay varias técnicas para escribir los requerimentos la más usada es Historia de Usuario, y otra que se usaba anteriormente es Caso de Uso.

Primero se tiene una idea gral. De lo que tiene que tener el sistema luego los detalles se negocian en las conversaciones, los PBIs se detallan a medida que se necesitan, cada PBI representa un valor de negocio que en algún momento nos van a contar los detalles.

Al comienzo los PBIs son grandes tienen poco detalle luego se van refinando se van detallando cada vez más a través de conversaciones con el PO y los StakeHolders.

Llega un, momento que estos PBIs son lo suficientemente chicos como para colocarlos en un sprint. No todos los PBIs tiene que tener el mismo nivel de detalles.

Scrum NO especifica ningún formato para los PBIs, entonces se usa gralmente historias de Usuario ( Users Stories ).

**User Stories:**

1. Es una forma de expresar PBIs.
2. Es una descripción corta contada desde la perspectiva de la persona que desea la nueva capacidad.
3. Es entendible para gente que no es técnica.
4. Describe QUE es lo que se debe implementar.

Una forma de contar las user Histories es utilizando las 3 C ( card-confirmation-conversation )

Conversation: A través de conversaciones entre el PO, Stakeholders y DT se comunican los detalles. La User Storie es una promesa de que se va a tener esa conversación.

En la tarjeta ( card ) se escribe en la parte de adelante el siguiente template

**Como <rol de usuario > quiero < objetivo > para < beneficio >**

Confirmation:

En la parte de atrás de la tarjeta se escriben los **Criterios de aceptación,** son los detalles, son condiciones de aceptación que nos ayuda a clarificar el comportamiento deseado del sistema por parte del cliente.

Una User story se puede refinar, es decir se puede detallar aún más, por ejm :

* **Epica** : es una user Storie que no sabemos cuanto tiempo nos va a llevar realizarla. Se puede refinar.
* **Fatures**: son User Stories que se pueden estimarcuanto tiempo nos lleva pero son muy grandes como para colocarlas dentro de un sprint.
* **Sprintable**: son User Stories que entran en un sprint.

Epica, Fatures, Sprintable responden a **qué** es lo que tenemos que hacer

**Tareas**: son el refinamiento final de una User Storie ( van al **cómo** lo voy a hacer)

Las User Stories son responsabilidad del PO.

Todos los involucrados pueden escribir User Stories ( Scrum Team + Stakeholders ), normalmente se escriben en el Warm-Up.

Product Backlog

Para escribir User Stories de buena calidad se utiliza el criterio INVEST

* Independiente: las User deben ser independientes entre ellas para que no nos obligue a tener un orden de como implementarlas.
* Negociable: La User es una promesa, un valor de negocio tiene un beneficio
* Estimable: se sabe cuanto tiempo lleva realizarla
* Small: es chica , bien detallada, normalmente se puede pasar a Done en 3 o 4 dias.
* Testeable: se puede verificar que la User va a cumplir con lo que el cliente quiere
* **ES MUY IMPORTANTE NO PERDER INFORMACION**
* **SEGUIR EL TEMPLATE**
* **AGREGAR DETALLES EN LOS CRITERIOS DE ACEPTACION**
* **SE REFINA EN 2 O MAS USERS STORIES**

El User Story Mapping ( USM ) nos ayuda a organizar las Users Stories y nos permite verlas en un contexto. En su confección participan el Scrum Team + StakeHolders, en gral se hace en el Warm-Up

Pasos para crear el USM

1. Identificar los objetivos
2. Construir el backbone
3. Identificar y agrupar las Users Stories
4. Agregar Stories, Users, tareas, faltantes
5. Priorizar las User Stories
6. Dividir las Users en releases

Hay 4 Niveles:

1. Nivel 1 están los roles ( los usuarios del sistema ).
2. Nivel 2 están las User Activities, son las principales actividades que hace el usuario con el sistema.
3. Nivel 3: esta el backbone, es la columna vertebral del diagrama, son features de alto nivel, en gral. son las épicas.
4. Nivel 4: están las Users stories, están divididas por distintas releases, cada reléase nos dice cuales son las Users más prioritarias.

En cada reléase se define el MVP ( mínimo producto viable )

**UML ( Lenguaje de Modelamiento Unificado )**

**UML significa Lenguaje de Modelamiento Unificad,o es un lenguaje de modelado visual que se usa para especificar, visualizar, documentar y construir artefactos de un sistema de software. Está pensado para usarse independientemente de la metodología, provee un conjunto de diagramas que pueden ser utilizados para documentar diferentes aspectos del sistema.**

**Cuando se desarrolla un sistema y hay que documentar cierto aspecto del mismo se acude a UML y nos fijamos que diagrama puede satisfacer esa necesidad de modelamiento.**

**UML no es una metodología, no me dice que tengo que utilizar los diagramas en un orden determinado, lo puedo utilizar en cualquier metodología.**

**Surge de la premisa de que un gráfico me da más información que una descripción escrita en palabras.**

UML tiene diagramas estructurales o estáticos y los comportamentales o dinámicos

**diagramas estructurales o estáticos**

1. diagrama de clases: modela las clases del sistema con sus atributos y relaciones
2. diagrama de objetos: modela los objetos en un momento particular, es una instantánea en un momento en particular, es una foto instantánea
3. diagrama de componentes: modela los componentes físicos del sistema, cuáles son los html, las librerías, los ejecutables, del sistema
4. diagrama de Deployment o despliegue: modela la topología del hardware, cuáles son los nodos y como se comunican entre ellos

**diagramas comportamentales o dinámicos**

1. **diagrama de casos de uso :** permite modelar cual es la funcionalidad del sistema, cuales son los limites del sistema y quienes están involucrados en el sistema.
2. diagrama de secuencia: Describe el orden en que se envían los mensajes entre los objetos, es igual que el de colaboración pero se dibuja distinto
3. diagrama de colaboración: describe que objetos están participando , , es igual que el de secuencia pero se dibuja distinto.
4. diagrama de transición de estados: Modela cuando un objeto estáen distintos estados, ejm una red puede estar desconectada, activa, sin conexión, transfiriendo datos.
5. diagrama de actividades: modelan las actividades que el sistema debería realizar, es un complemento del diagrama de casos de uso.

**CASOS DE USO**

Un diagrama de caso de uso me permite representar, la funcionalidad que va a tener el sistema y cuál es el límite del sistema. Representa la información que va a recibir el sistema del exterior, que información se va a generar y que cosas va a tener que hacer el sistema para cumplir con una funcionalidad, muestra que cosas están afuera y cuales están adentro.

Casos de uso es una forma de especificar los PBIs dentro de SCRUM. Casos de Uso es un poca más rígido que Users Stories.

Para modelar tengo:

**El caso de uso**: Se dibuja con una elipse y lleva como nombre un verbo en infinitivo

**El actor** : representa el rol que esta jugando la persona que interactúa con el sistema, se dibuja con un hombrecito y el nombre de ese rol. Una persona o actor puede representar diferentes roles. Los actores pueden ser personas, un dispositivo ( por ejm un sensor ), u otro sistema.

El actor: puede ser primario o secundario, por caso de uso tengo 1 actor primario y 0 o más actores secundarios. También está el actor reloj que activa el sistema en un tiempo particular

**Activación:** es una línea que une el caso de uso con el rol, el caso de uso se activa por un rol.

**Relaciones:** entre casos de uso, son flechas, nos permite reestructurar de manera mas clara las relaciones entre casos de uso.

El caso de uso se complementa con un template de documentación llamado Descripción del caso de uso. Este template tiene:

1. El nombre de caso de Uso.
2. Descripción breve de los que hace el caso de uso.
3. Actor primario/secundarios: Actores involucrados en el caso de uso.
4. Trigger : es el disparador de la activación.
5. Flujo de ejecución/Curso básico.
6. Cursos alternativos. Son desviaciones del curso básico.
7. Precondiciones: algo que es verdadero antes de desarrollar la funcionalidad.
8. Postcondiciones: es algo que es verdadero después de desarrollar la funcionalidad.
9. Suposiciones: son verificaciones que ya están comprobadas.
10. Casos de uso que extiende: casos de uso a los que les da nueva funcionalidad en situaciones particulares.
11. Casos de uso incluidos: son casos de uso que necesita para completar su funcionalidad
12. Finalización del caso de uso: situaciones por el cual el caso de uso termina.

RELACIONES DE CASOS DE USO

**GENERALIZACIÓN**

UML trata de aplicar cuando corresponde, conceptos de prog. Orientada a objetos, generalización es el equivalente a herencia, cuando un caso de uso padre va a hacer heredar su comportamiento a los casos de uso hijos. Se dibuja con una flecha

**INCLUSIÓN**

Cuando dos o más casos de uso comparten una parte del comportamiento, el

comportamiento compartido se factoriza en un nuevo caso de uso

El nuevo caso de uso no es un caso de uso normal, es decir, no será

activado por un actor (90%)

Cuando se activa el caso de uso original, los casos de uso incluidos

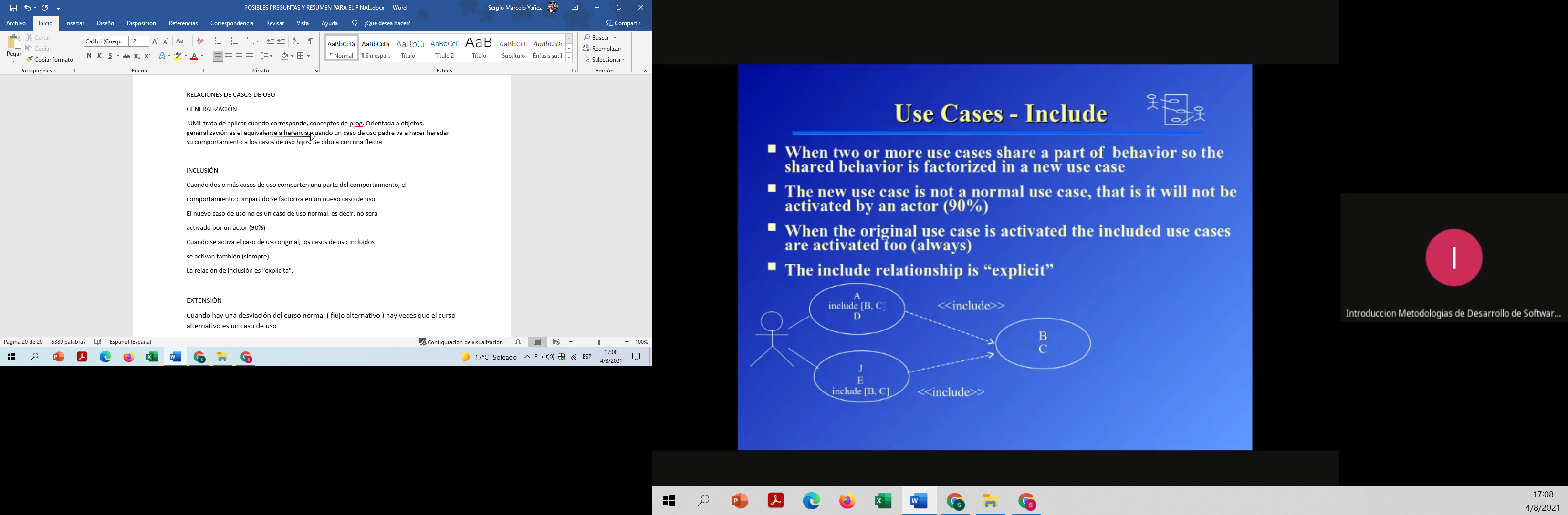
se activan también (siempre)

Las inclusiones deben ser incluidas por al menos 2 casos de uso al menos que sea caso de uso en si mismo( es decir un actor activa 2 casos de uso y a la vez uno incluye al otro)

La relación de **inclusión es "explícita**" porque en la descripción del caso de uso originario pongo la palabra INCLUDE y el nombre del caso de uso incluido. Cuando se activa el caso de uso originario se activa SIEMPRE el incluido, la flecha apunta al caso de uso incluido, y el caso de uso incluido no es activado por actores en la mayoría de los casos.

En la descripción del caso de uso original en la parte de casos de uso incluido se coloca el nombre del caso de uso incluido

La descripción del caso de uso incluido se realiza como la de un caso de uso común



**EXTENSIÓN**

La relación de extensión se da cuando el caso de uso tiene un problema y la solución se la da otro caso de uso. Un caso de uso ayuda al otro.

Se relacionan por una flecha punteada que apunta para el lado del caso de uso que tiene el problema y en la flecha se tiene que colocar la condición, el que tiene el problema no sabe que problema tiene ni quien se lo va a solucionar.

Cuando se activa el caso de uso original, los casos de uso que lo extienden se van a activar cuando se cumple una condición y el caso de uso que extiende se mete a solucionar el problema

El nuevo caso de uso es un caso de uso normal, es decir, es activado por un actor (90%)

La relación de **extensión es "implícita".**

En la descripción del caso de uso original se coloca el nombre del punto de extensión o condición

La descripción del caso de uso que extiende se hace como uno normal, pero se completa el item de casos de uso extendido, diciendo a que caso de uso va a ayudar y cuándo lo va a ayudar.

**DIAGRAMA DE CLASES**

Se utilizan para:

• Explorar conceptos del dominio

• **Analizar requerimientos**

• Mostrar el diseño detallado de software orientado a objetos

Generalmente contiene:

• Clases

• Interfaces

• Relaciones

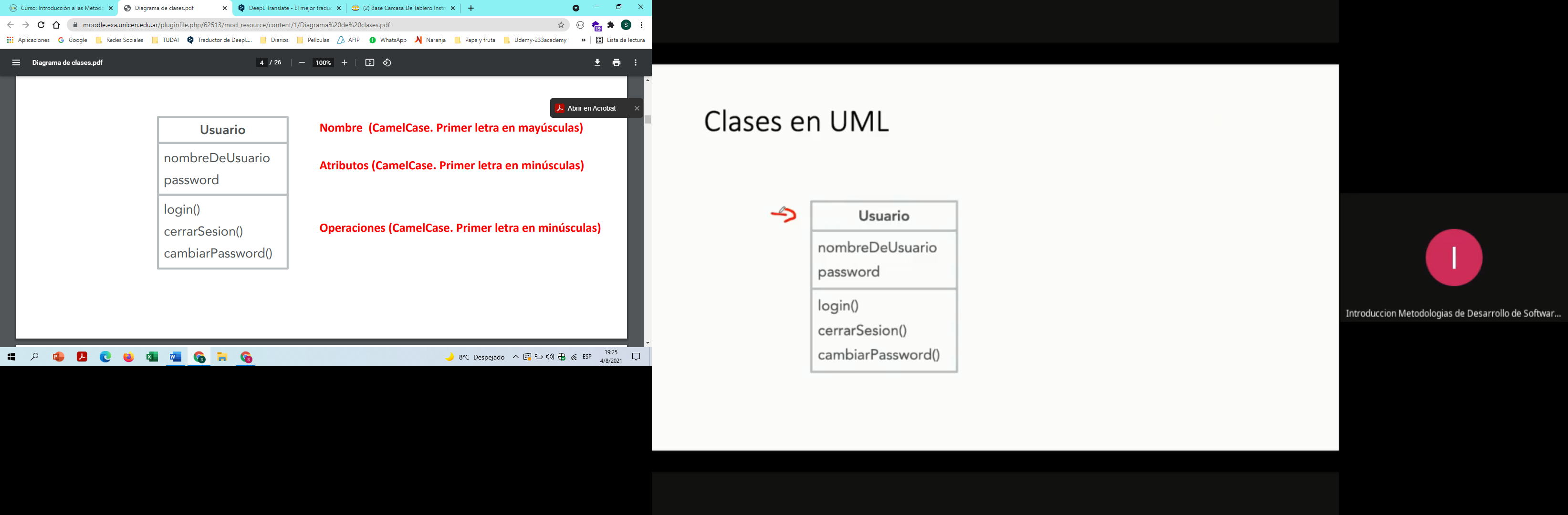
Una clase es la descripción de un conjunto de objetos que comparten los mismos:

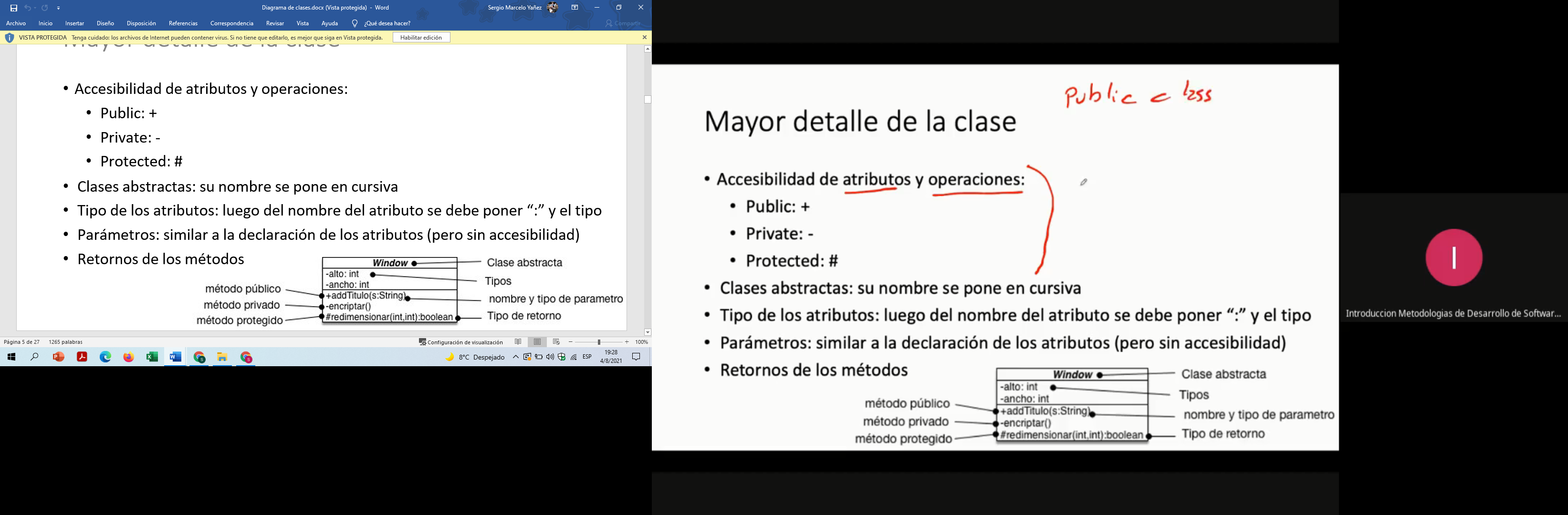
• Atributos

• Operaciones

• Semántica

•Las clases capturan el vocabulario del sistema que se está desarrollando

• Representan objetos del mundo real y elementos conceptuales (ej. Conversacion, politica de ordenamiento, etc.) 



**Relaciones entre clases**

Las relaciones entre clases modelan la colaboración entre objetos.

Dos clases A y B están relacionadas si:

• Un objeto de la clase A envía un mensaje a un objeto de la clase B

• Un objeto de la clase A crea un objeto de la clase B

• Un objeto de la clase A tiene un atributo cuyo tipo es B o que es una colección de objetos de tipo B

• Un objeto de la clase A recibe un mensaje con un objeto de la clase B como parámetro

• La clase A es superclase de B

• La clase A implementa la clase B

Tipos:

Generalización ( herencia): Indican una relación entre una clase gral. que es la superclase (padre) y un tipo más específico que es la subclase ( hijo) ( flecha continua con triangulo vacío )

Realización: Es lo mismo que una herencia pero para interfases, se representa una linea punteada con triangulo vacío

Asociación: Es una relación es estructural, si al menos una clase tiene un atributo del tipo de otra clase entonces la relación es de asociación. Se muestra como una línea llena con adornos ( navegabilidad que es una flecha indicando el sentido en que se envía los mensajes, nombre o rol , multiplicidad ) .

Modela conexiones del tipo: “ tiene” , “es de “ , “ conoce “.

Agregación: Es un tipo especial ( subtipo ) de asociación, la agregación especializa a la asociación una clase representa el todo y las otras representan las partes. Modela conexiones del tipo “ Esta formado por “. Se dibuja con una línea y un rombo vacío en la clase que representa el todo.

Composición: es un subtipo de la agregación, el tiempo de vida de las partes están ligadas al todo.

Se tiene que cumplir que:

* la clase tiene un atributo del tipo de la otra
* La clase está formada por la otra o
* Cuando se destruye el todo se destruyen las partes
* Se dibuja una línea llena con un rombo lleno y llevan las multiplicidades.

Clase de asociación: Es cuando tenemos una relación de asociación con multiplicidad N a N ( muchos a muchos). Se dibuja con una línea punteada.

**Las relaciones de agregación y composición sólo pueden inferirse si hay un contexto sino es de asociación.**

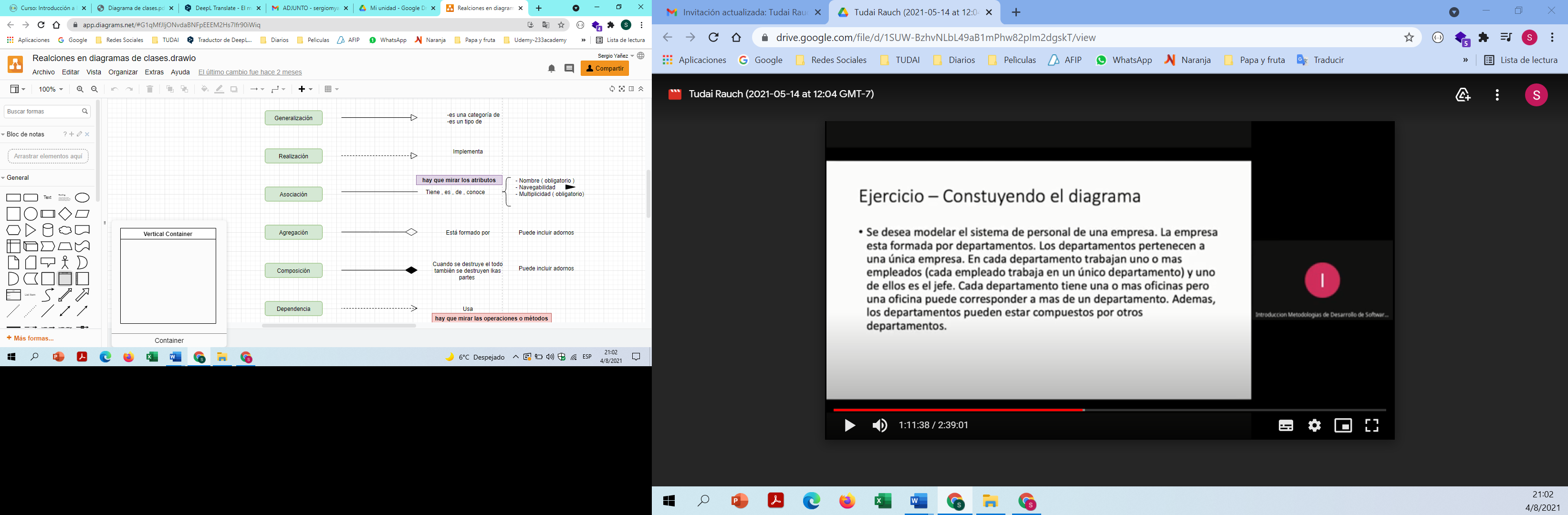
**Sólo las relaciones estructurales ( asociación, agregación, composición ) llevan adornos**

Dependencia: Es la relación más débil sólo se muestra si no hay otro tipo de relación, esta relación existe cuando:

* Una operación del objeto de la clase A define una variable cuyo tipo es B o que es una colección de objetos de tipo B
* Un objeto de la clase A envía un mensaje a un objeto de la clase B
* Un objeto de la clase A crea un objeto de la clase B
* Un objeto de la clase A recibe un mensaje con un objeto de la clase B como parámetro
* Una clase importa otra clase.

Modela conexiones del tipo “ usa “ , se representa con una línea punteada

**Cuando me dan un código no tengo contexto que se pueda chequear, entonces no voy a poder saber si la relación es de agregación o composición, todas las estructurales van a ser de asociación.**



**DIAGRAMA DE SECUENCIA**

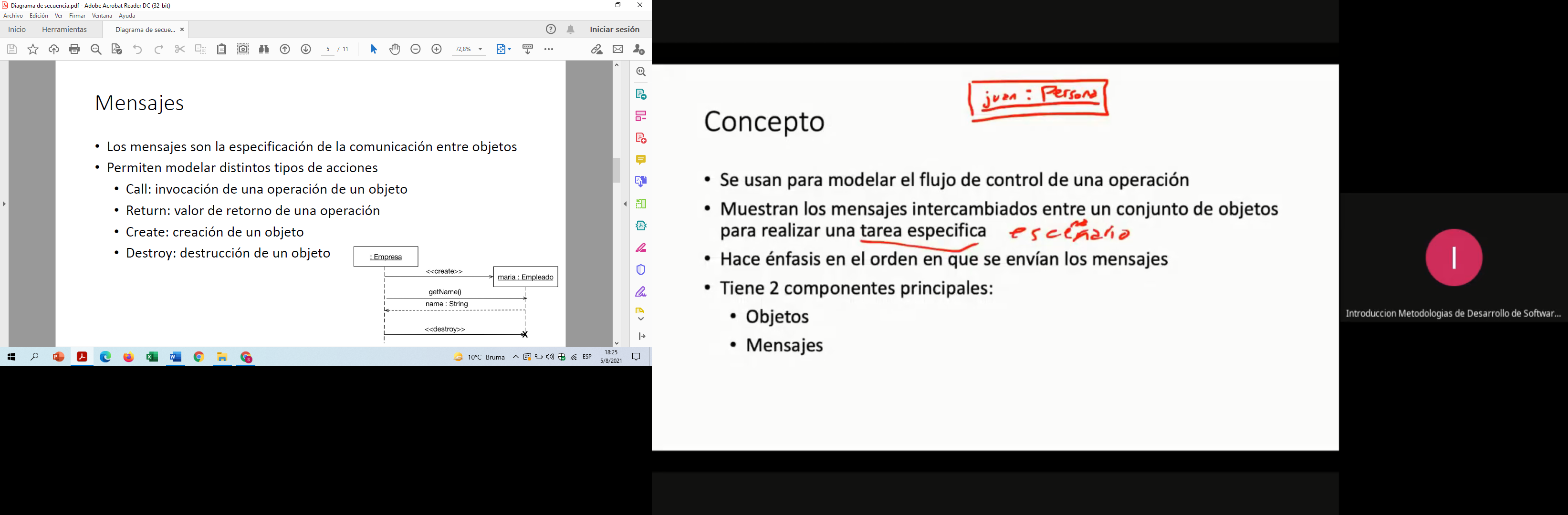
UML presenta unos diagramas de Interacción:

* Diagrama de secuencia hace énfasis en el órden en que se envían los mensajes.
* Diagrama de colaboración hace énfasis entre que objetos se dan los mensajes

Los diagramas de secuencia se utilizan para mostrar cómo funcionan determinadas partes del sistema , en vez de acudir al código, que lleva más tiempo, en un diagrama de secuencia se ve mas rápido y sencillo.

* Sirven para mostrar como se envían los mensajes entre los objetos en tiempo de ejecución cuando realizan una tarea específica ( en un escenario determinado ).
* Hace énfasis en el órden en que se envían los mensajes.
* Tiene 2 componentes:
  + - Objetos
    - Mensajes



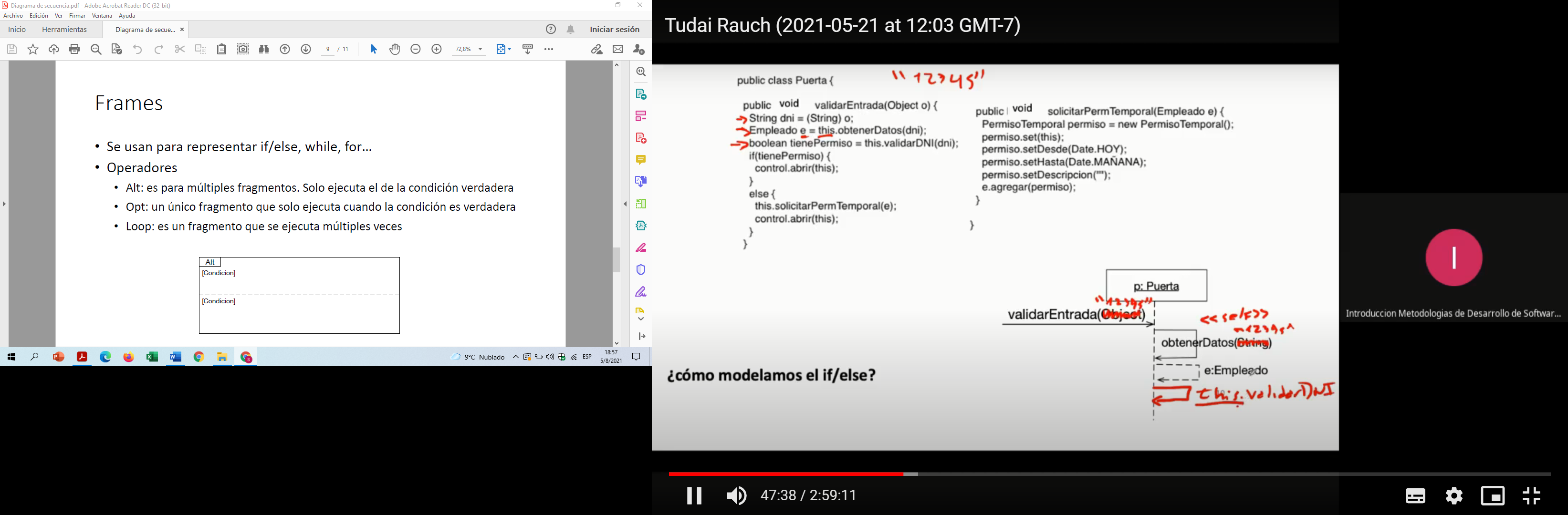


Call se representa por una línea llena que va de la línea de tiempo de vida del objeto que llama a la línea de tiempo del objeto que es llamado, y lleva el nombre del llamado o método que se utiliza.

Return se representa con una línea punteada desde el objeto llamado apuntando hacia el objeto que llamó, y se coloca sobre la línea el nombre y tipo de lo que devuelve.

Create es cuando un objeto crea otro ( es el new ), el objeto creado se dibuja un poco más abajo del objeto que lo crea , se muestra con una línea llena desde la línea de vida del objeto creador apuntando al objeto creado y se le pone la palabra créate entre corchetes angulares.

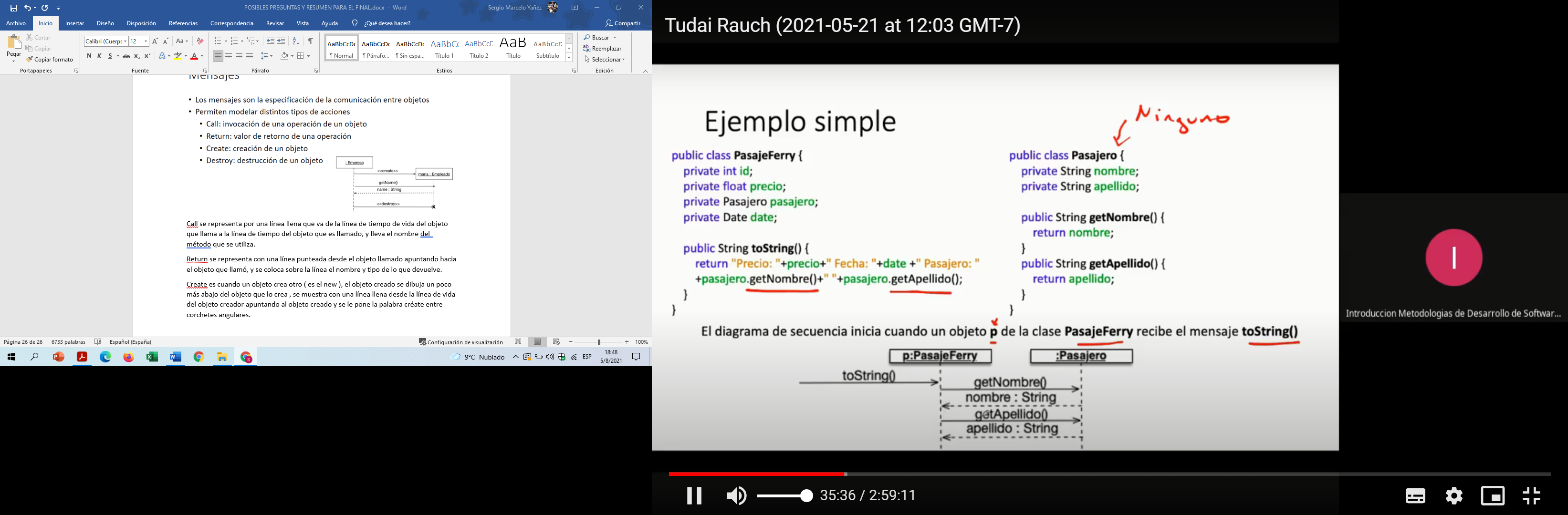
Un elemento del diagrama de secuencia es el Frame



**Alt:** se utiliza para un if/else o un case

**Opt:** se utiliza cuando hay un if solo, hay una sola condición

**Loop**: se utiliza con un while, for, foreach, tiene una sola condición que es por la cual cicla ese loop



Otro ejemplo:

