UCAB UNIVERSIDAD CATÓLICA ANDRÉS BELLO

GESTIÓN DE PRUEBAS

Gestión de pruebas por María Angélica Pérez de Ovalles. UCAB, está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

OBJETIVOS

Comprender los conceptos asociados a la disciplina Gestión de pruebas.

Identificar los pasos a seguir, según MECAP, para formular casos de pruebas funcionales, bajo el enfoque clásico o el ágil.

TABLA DE CONTENIDO

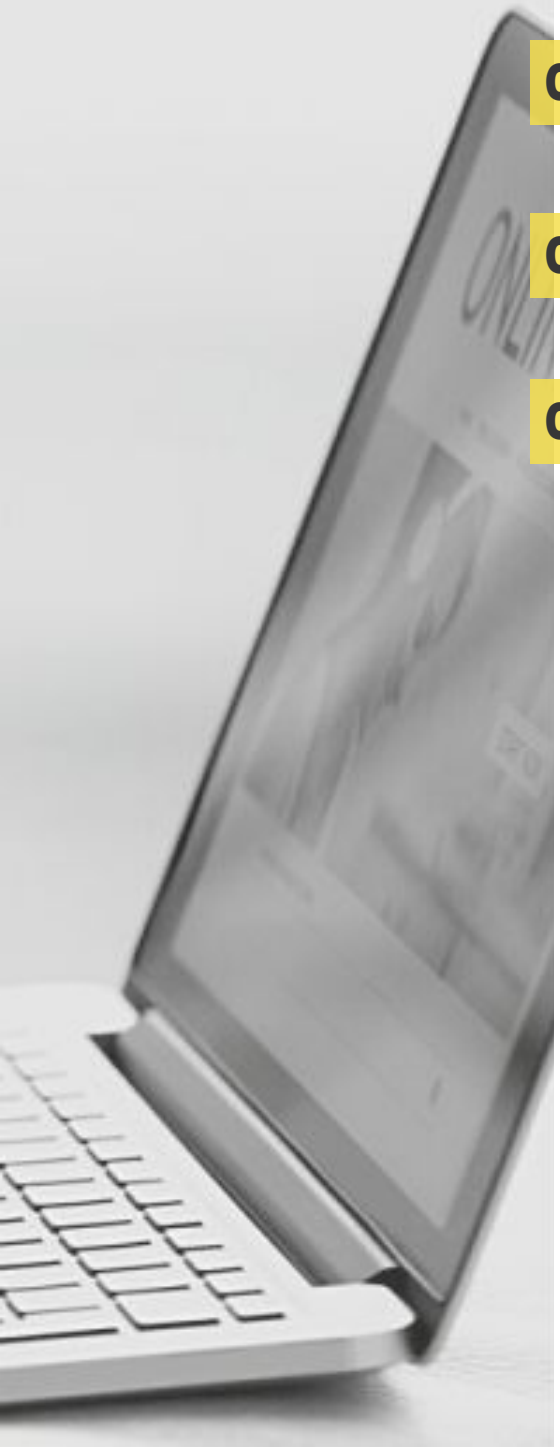
01 Aspectos de la prueba

02 Prueba de *software*

03 Estrategia de prueba

04 Enfoque de pruebas

05 MECAP



¡La calidad! ¡Un verdadero dolor de cabeza en el mundo de la ingeniería de software! Nunca hay tiempo para realizar las pruebas y cada vez tenemos usuarios más **exigentes**. Es el elemento diferenciador, es lo que decide tu reputación. Ya el problema no es encontrar una app que haga lo que deseas, sino la de mejor reputación. Y esa es la que tiene mejor calidad.

Ahora bien, la calidad solo se logra si hay una verificación continua de ella, si se le da la importancia que tiene, asignándole los recursos necesarios. La literatura sobre el tópico de pruebas es amplia, variada y muy confusa. Probar es hacer Control de Calidad, es garantizar que el producto que se genera está libre de defectos. Aquí vamos a aclarar estos conceptos.

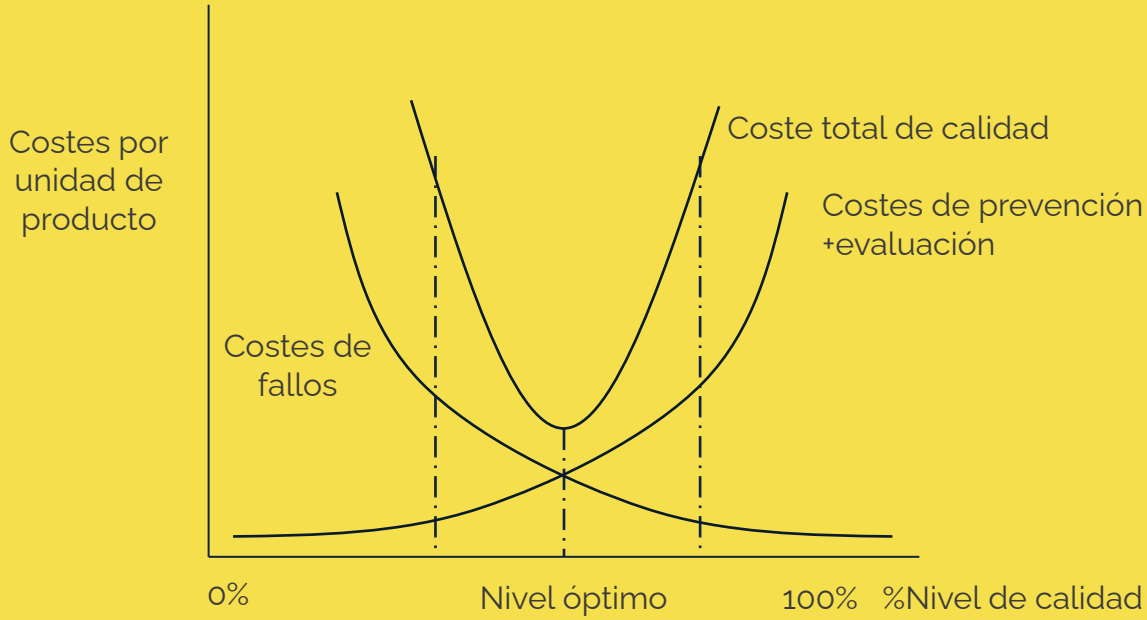
Por otro lado, los sistemas actuales son tan complejos que es muy probable que no podamos probarlo todo. Por ello va a ser necesario definir una **estrategia de prueba** y enfocarnos en algo preciso; estos son dos tópicos de este tema, cerrando con la propuesta de un método para formular Casos de Prueba, a partir de Casos de Uso, llamado MECAP.

01 Aspectos de la prueba

Si hacemos memoria, recordaremos que, en la primera sesión, conversamos sobre las **mejoras** prácticas. Una de ellas es la verificación continua de la calidad. Una de las estrategias más usadas para garantizar la calidad en el *software* son las pruebas.

Por razones éticas debemos entregar un producto probado y que cumpla con las **especificaciones**. ¿Qué ponemos en riesgo si no probamos o si el proceso de pruebas no es confiable? ¿Cuánto dinero debemos invertir en probar un *software*?

Si observas la figura, hay un punto en el que no tiene "mucho sentido" seguir probando, pues se incrementa demasiado el costo del producto frente al número de defectos encontrados.



Zona de mejora
Costes de fallos > 70%
Costes de proyección <10%

Zona de indiferencia
Costes de fallos ~ 50%
Costes de proyección ~ 10%

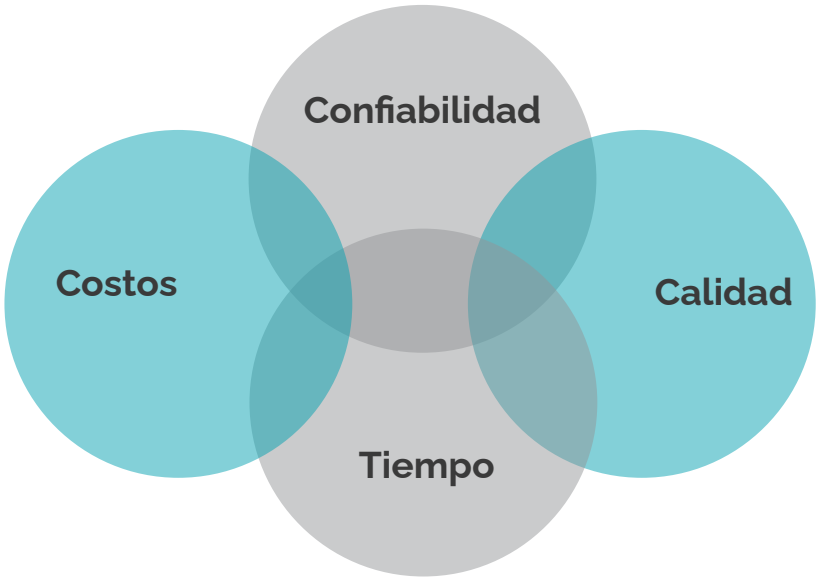
Zona de perfeccionamiento
Costes de fallos < 40%
Costes de proyección > 50%

Fuente: Aspectos de la prueba. Extraída de Lopez Fojo (2012)

Por lo que el fabricante decide no probar más y asumir las consecuencias de cualquier fallo. Por supuesto, esto tiene sus excepciones. Por ejemplo, en el caso de un *software* que monitorea un avión de pasajeros, ¿podemos asumir las consecuencias de un fallo?

Por lo que debemos hacer un equilibrio entre los **aspectos** a considerar cuando hay que decidir cuánto probar. El análisis de este equilibrio debe hacerse antes de iniciar el proceso de desarrollo del *software*.

Los aspectos a considerar son:



Fuente: Aspectos a considerar antes de iniciar software.

A Confiabilidad:

- Se traduce en: ¿qué tan **confiable** es la prueba y los resultados que esta arroja? Dependiendo del tipo de *software* hay que considerar niveles y tipos de pruebas que permitan señalar la confiabilidad de las pruebas a realizar o ya hechas.

- Un ejemplo de confiabilidad es la prueba del embarazo, de las que venden en farmacia. Si da positivo seguro tendrás un bebe, sino no es seguro que no lo estés.

B Costos:

- Las pruebas presentan entre el 30 % y 50 % de los costos de software. Esto quiere decir que si estimas que el proyecto puede durar tres meses, ¡al menos dos serán para probarlo!
- Las pruebas son **difíciles**.
- No se utiliza una metodología clara. Depende de las habilidades de los individuos. Utilizando **estrategias** y herramientas apropiadas se puede mejorar la productividad y la efectividad de las pruebas del *software*.
- ¿La organización cuenta con un ambiente de pruebas?

C Tiempo:

- La ejecución de las pruebas se encuentra dentro de una **planificación**, ya que toman mucho tiempo y no se pueden dejar al azar cuando se realizan.
- Se chequean los objetivos asociados a las pruebas para determinar si se pasa a la siguiente iteración. ¿Cuándo es más recomendable realizar las pruebas y proveer el feedback?

D Calidad:

- Significa **ausencia** de defectos. Ajuste a los propósitos.
- Un *software* puede estar libre de defectos, pero no hace lo que se quiere (funcionalidad). El objetivo de las pruebas es **evaluar** la calidad del producto final, evaluar la calidad de los componentes y la arquitectura que da forma a estos componentes.

Cerremos entonces este punto. Cuando se inicia el proceso de desarrollo, se reúne el equipo y se analizan estos cuatro aspectos de la prueba para que los probadores, con base a su análisis, puedan definir una **estrategia** y un enfoque de prueba.

02 Prueba del *software*

La prueba del *software* implica ejecutar o inspeccionar la implementación de un sistema con datos de pruebas.

Se **examinan** las salidas y su entorno operacional para comprobar que funciona tal y como se requiere.

La prueba tiene por objetivo demostrar que un software cumple su propósito y descubrir sus **defectos**. Busca defectos o anomalías en su funcionamiento. Hay dos vías para identificar anomalías o defectos:

1. **Verificación:** ¿estamos construyendo el producto correctamente? Son actividades que aseguran que el *software* implementa correctamente una función específica.

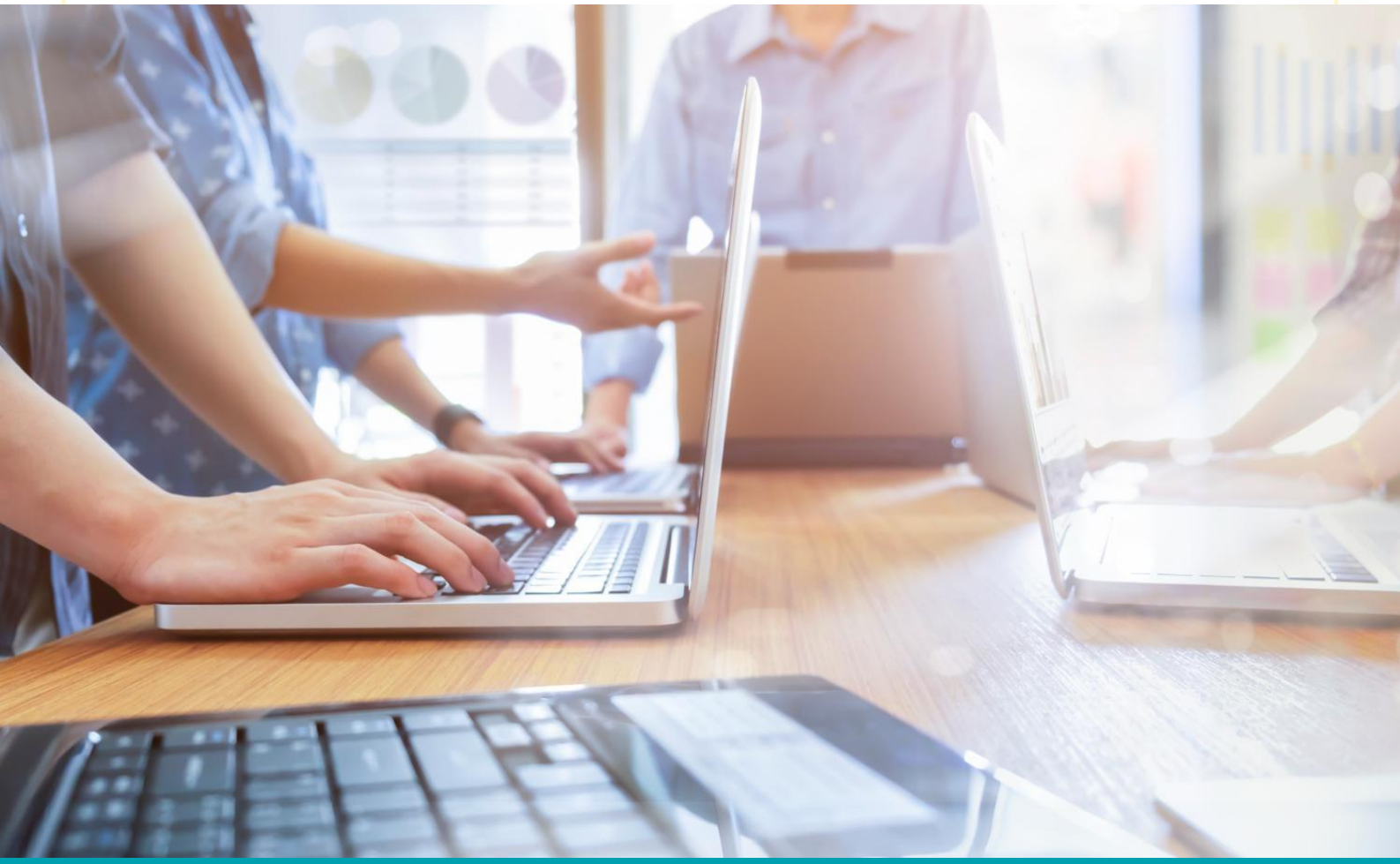
2. **Validación**: ¿estamos construyendo el producto correcto? Son actividades que aseguran que el software construido se ajusta a los requisitos del cliente.

Es oportuno aclarar tres conceptos asociados con la prueba:

- **Error (error):** es una **equivocación** cometida por un desarrollador. Un error sintáctico, una equivocada interpretación de un requerimiento o de la funcionalidad de un método.

- **Defecto (*fault, defect*):** un error puede conducir a uno o más defectos. Un defecto se encuentra en un artefacto y puede definirse como una diferencia entre la versión correcta del artefacto y una versión incorrecta.

- **Falla (*failure*)**: en terminología IEEE, una falla es la discrepancia visible que se produce al **ejecutar** un programa con un defecto.



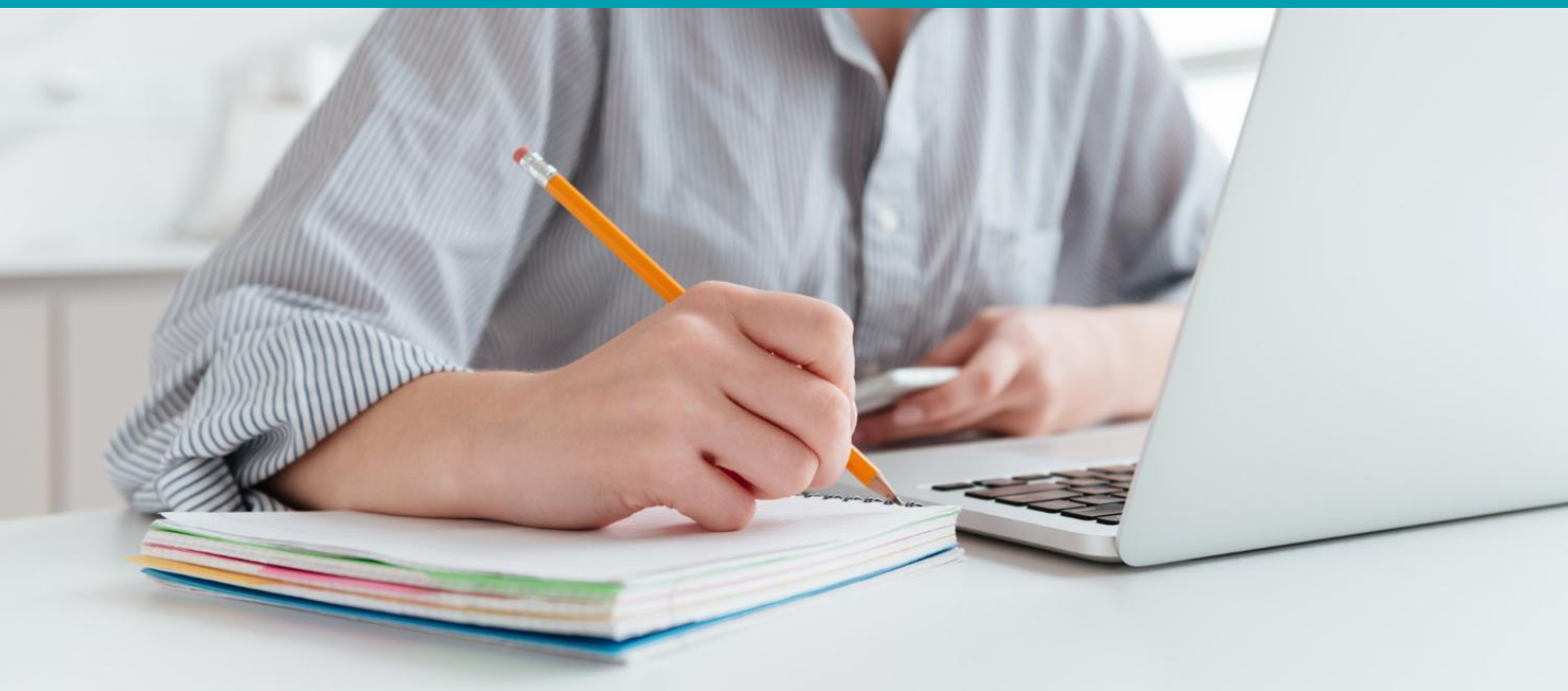
Entonces, para encontrar los defectos que se introducen al desarrollar un *software* complejo hay que formular una estrategia.

03 Estrategia de prueba

La **estrategia** de prueba nos indica qué, cómo, cuándo y dónde se debe probar, a través de un plan de acción que nos permita cumplir un objetivo. Es decir, hay que hacer un plan de pruebas

¿qué?	el componente gestión de productos.
¿como?	haciendo pruebas de patrón, estrés y funcionalidad.
¿cuándo?	dos ciclos en el segundo *sprint*.
¿dónde?	en la máquina de Juan

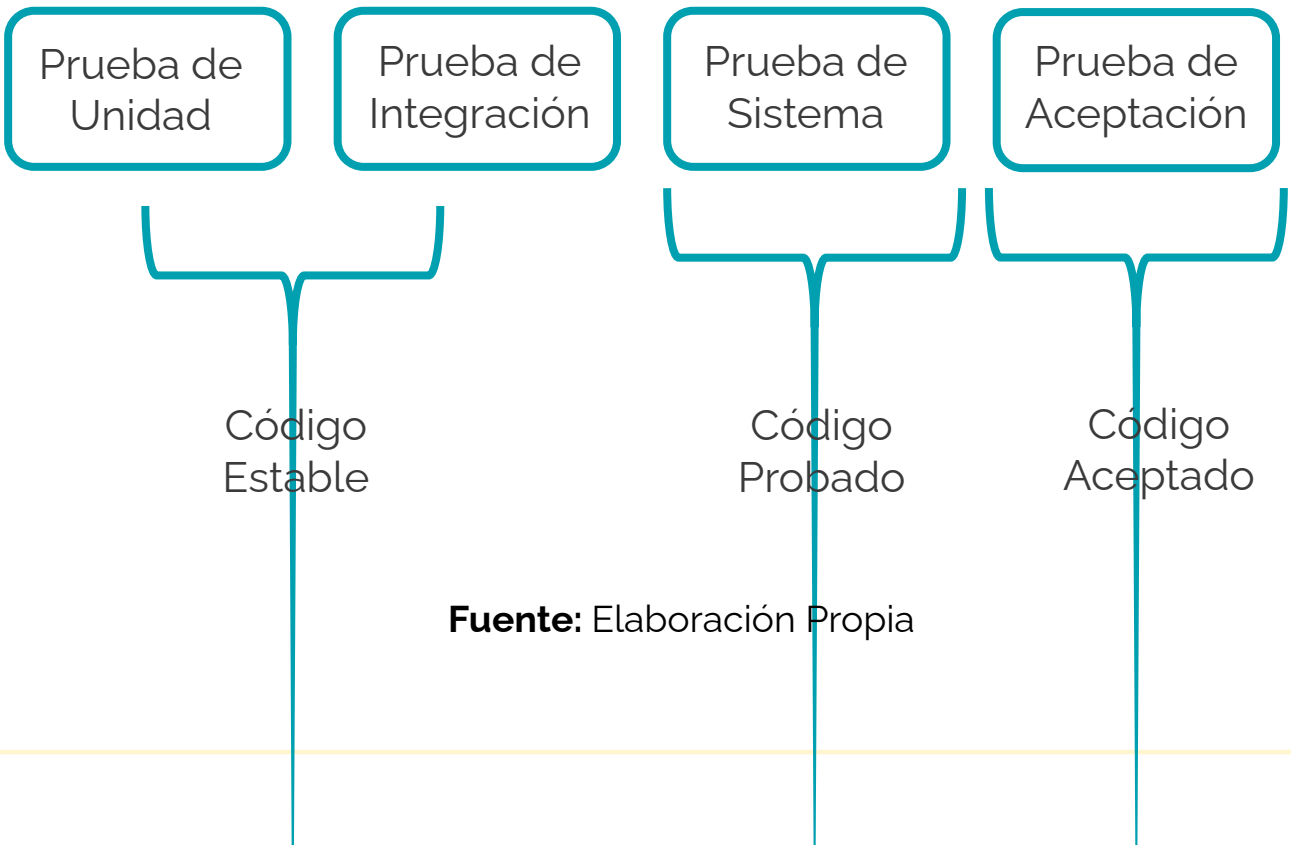
El cómo tiene sus complicaciones, no es tan fácil de formular. Por ello se recurre al concepto de enfoque de prueba.



04 Enfoque de pruebas

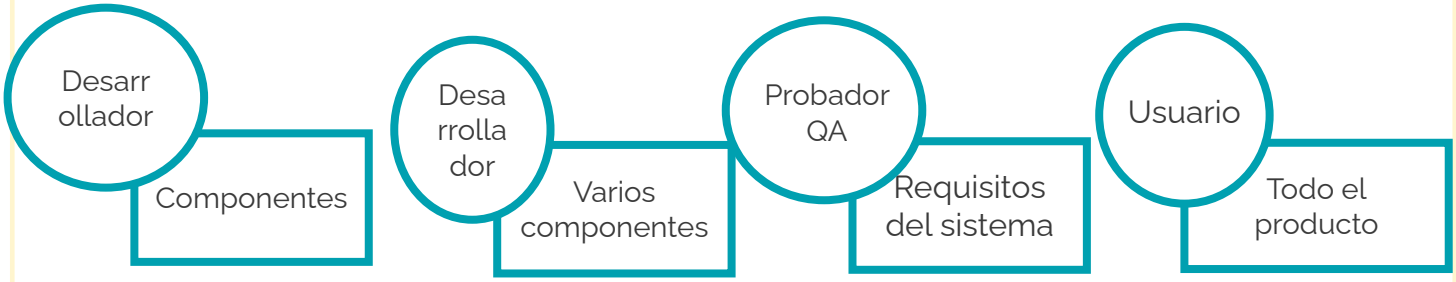
Para entender mejor qué es un enfoque debemos precisar más aún otro conjunto de conceptos que están muy confusos en la literatura. Luego de años de estudio e investigación, se analizaron estos conceptos con mis colegas y a continuación propongo esta organización de ellos, la cual llamaremos Dimensiones de la prueba. Ellas son:

1 **Nivel de la prueba:** pueden ser cuatro: de unidad (o de componente), de integración (de varios componentes), de sistema (o pruebas de requerimientos) y de aceptación (o de toda la app). Y estos dependerán del ambiente (máquina) en la cual se están realizando las pruebas. Cada nivel arroja un tipo de código. Cuando se terminan las pruebas de unidad y de integración, que son realizadas en el ambiente de desarrollo, el código obtenido es un código estable. Cuando se terminan las pruebas de sistema, que son realizadas en el ambiente de pruebas, se obtiene un código probado. Y cuando se terminan las pruebas de aceptación, que son realizadas en la máquina donde finalmente residirá el *software*, se le conoce como ambiente de producción, ya que se obtiene un código aceptado (figura 3).



Fuente: Elaboración Propia

Típicamente estos niveles de prueba son ejecutados por personas y en máquinas diferentes (**figura 4**):



Fuente: Elaboración Propia

2 **Tipo de prueba:** existen múltiples tipos de pruebas. Solo se mencionan algunas de ellas:

- Prueba patrón (*Benchmark test*): determina el **funcionamiento** de un blanco-de-prueba contra un estándar del software o medidas existentes.
- Prueba de la configuración: determina cómo el blanco-de-prueba funciona en **diversas** configuraciones (*hardware* o *software*) durante la prueba.
- Prueba funcional: determina cómo las funciones del blanco-de-prueba ejecutan el caso de uso **requerido** según lo previsto (funcionalidad).
- Prueba de la instalación: determina cómo el blanco-de-prueba se instala en diversas configuraciones o bajo diversas condiciones, tales como espacio de disco restringido, velocidad del procesador, espacio de memoria, sistema operativo, etc.
- Prueba de la integridad: determina la confiabilidad, la robustez, y la resistencia de los blancos-de-pruebas, a través de las fallas durante la ejecución.

- Prueba de la carga (*Load Test*): determina la aceptabilidad del funcionamiento de blancos-de-prueba bajo varias **condiciones** operacionales, tales como número de usuarios, número de transacciones, y así sucesivamente, mientras que la configuración sigue siendo constante.

- Prueba de desempeño (*Performance*): determina la aceptabilidad del funcionamiento de los blancos-de-pruebas usando varias configuraciones, mientras que las condiciones operacionales siguen siendo constantes (mide eficiencia).

- Prueba de tensión (*Stress*): determina el funcionamiento de los blancos-de-pruebas, utilizando condiciones **anormales** o extremas; por ejemplo: recursos disminuidos o un número extremadamente alto de usuarios.

- Prueba de la regresión: es una estrategia donde pruebas previamente ejecutadas son nuevamente puestas en marcha a través de una nueva **versión** como blanco-de-prueba para asegurarse de que no ha sido perjudicada la calidad cuando se han agregado nuevas capacidades.

3 **Características de calidad** que se evalúan (inspirados en el ISO25010): Cada prueba, al aplicarse, debe estar buscando defectos relacionados con alguna de las características de calidad. Si estoy haciendo una prueba de carga puedo estar midiendo su eficiencia. Si estoy probando que efectivamente elimina los datos de un producto, estoy verificando su completitud funcional.

4 **Verifica o valida:** es decir, una prueba puede estar **comprobando**: si el producto está correctamente construido (verificar) si hace lo que se espera que haga (validar).

5 **Estática o dinámica**: si para hacer la prueba necesito ejecutar el código, es dinámica; si en cambio la hago sobre un diagrama, código etc. es estática.

6 **Manual o automática**: si la hace el probador/desarrollador es manual; si la hace un *software* es automática.

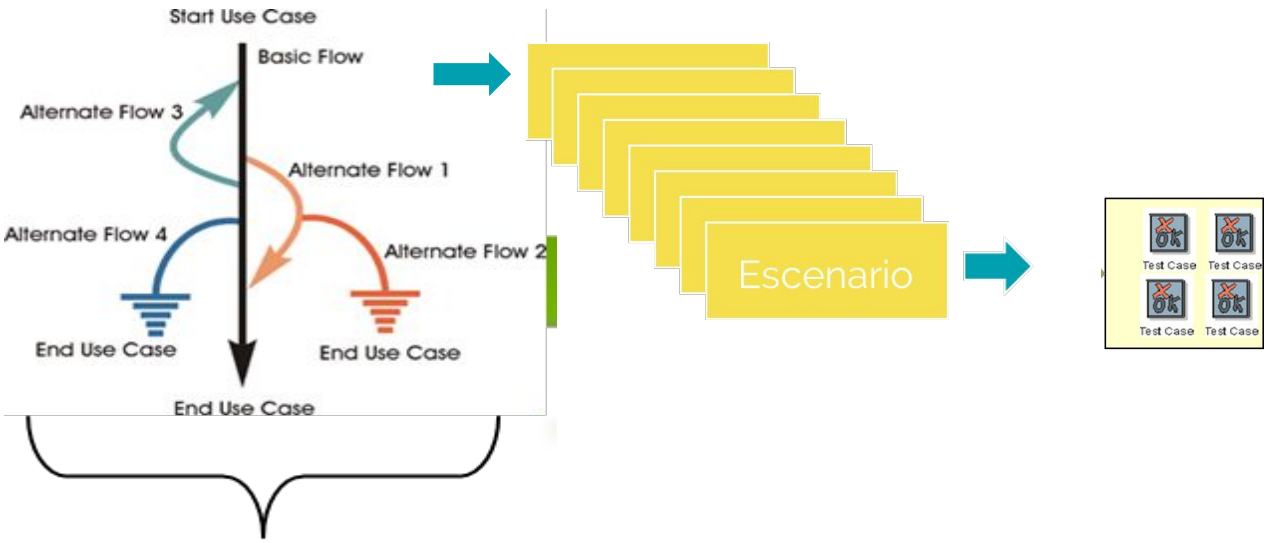
7 **Alfa o beta:** si el código no es estable es alfa; si es estable es beta.

Entonces, para definir un enfoque de prueba, debo decidir estas siete dimensiones. La forma en como el **subequipo** de pruebas presenta sus diferentes enfoques de prueba, varía de organización a organización. Lo importante es que se precise, tomando en cuenta todas estas dimensiones

05 MECAP

Uno de los elementos considerados en todos los estándares de calidad del proceso de desarrollo del software es la trazabilidad del proceso. Es decir, si la necesidad que va a resolver la app está asociada al control de los pagos de unos productos, la **funcionalidad** de la app debe contener capacidades que controlen los pagos de dichos productos. El diseño debe contener: capas, filtros, servicios, etc., que garanticen el control de los pagos de unos productos. ¿Y qué vamos a probar? Que la app controle los pagos de unos productos.

MECAP (Méndez, Pérez Mendoza 2007) es un método que permite formular Casos de rueba uncionales (es decir, evalúan la característica de alidad: la uncionalidad) a partir de sos de so. Este método surgió por una iniciativa del sector público venezolano y garantiza la trazabilidad entre casos de uso y casos de prueba (figura 5):



Fuente: MECAP. Extraída de Méndez, Pérez y Mendoza (2007)

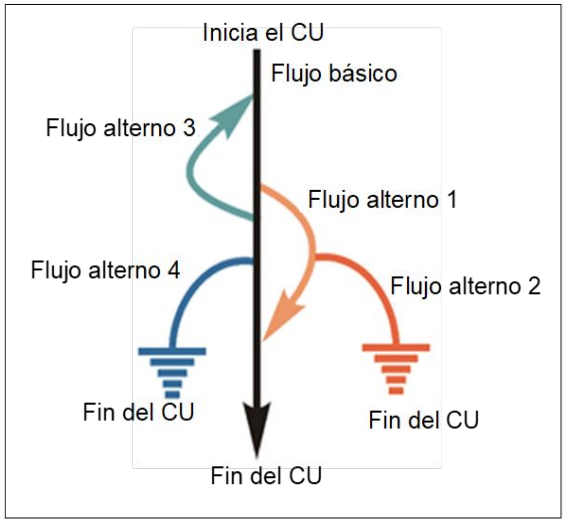
Consta de cuatro pasos:

Fuente: Pasos del MECAP. Extraída de Méndez, Pérez y Mendoza (2007)

Identificar escenarios

6 pasos

1.1 Identificar los escenarios tomando como base las narrativas
1.2 Presentar los escenarios tomando como base las narrativas de los CU
1.3 Chequear que se hayan representado gráficamente todos los FA



Identificar escenarios

6 pasos

1.4. Establecer todos los escenarios asociados a un CU a través de la tabla indicada.

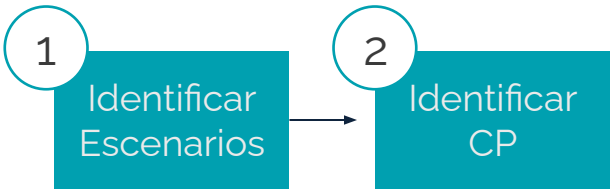
Nro. Escenario	Flujo Originario	Flujo Alterno	Próximo alterno	Próximo alterno
Escenario 1	Flujo Básico			
Escenario 2	Flujo Básico	Flujo alterno 1		
Escenario 3	Flujo Básico	Flujo alterno 1	Flujo alterno 2	
Escenario 4	Flujo Básico	Flujo alterno 3		
Escenario 5	Flujo Básico	Flujo alterno 3	Flujo alterno 1	
Escenario 6	Flujo Básico	Flujo alterno 3	Flujo alterno 1	Flujo alterno 2
Escenario 7	Flujo Básico	Flujo alterno 4		
Escenario 8	Flujo Básico	Flujo alterno 3	Flujo alterno 4	

1

Identificar Escenarios

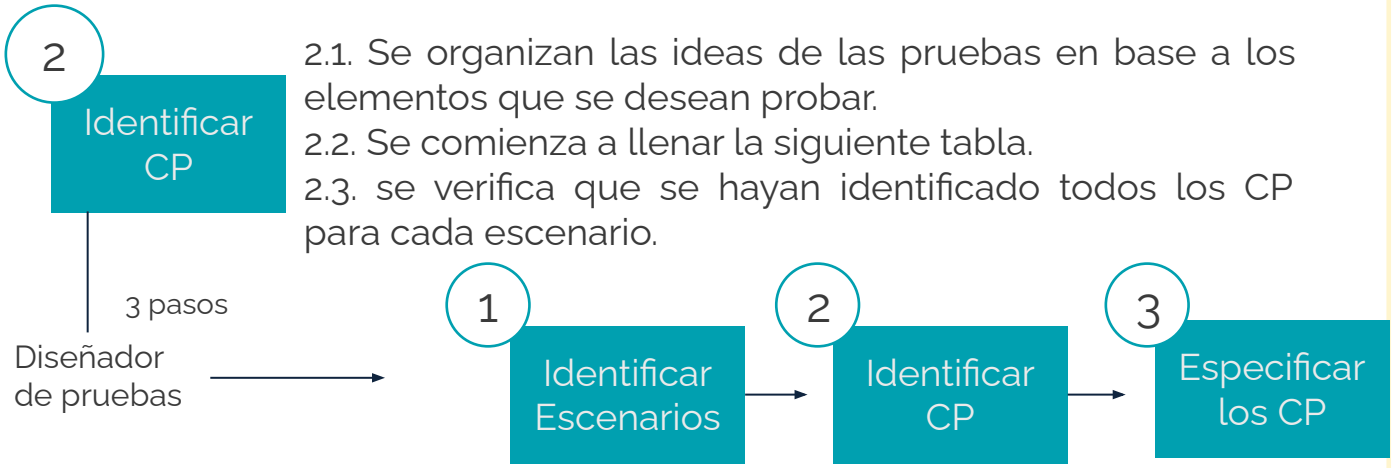
1.5. Identificar cada escenario del CU indicando el FB y/o Fas.
1.6. Verificar que se hayan identificado y descrito todos los escenarios posibles para cada CU.

6 pasos



Nro. Escenario	Flujo Originario	Flujo Alterno	Próximo alterno	Próximo alterno
Escenario 1	Flujo Básico			
Escenario 2	Flujo Básico	Flujo alterno 1: Error Password		
Escenario 3	Flujo Básico	Flujo alterno 1: Error Password	Flujo alterno 3: presionar cancelar	
Escenario 4	Flujo Básico	Flujo alterno 2: Error Password		
Escenario 5	Flujo Básico	Flujo alterno 2: Error Password	Flujo alterno 3: presionar cancelar	
Escenario 6	Flujo Básico	Flujo alterno 1: Error Password	Flujo alterno 2: Error Password	
Escenario 7	Flujo Básico	Flujo alterno 1: Error Password		Flujo alterno 3: presionar cancelar
Escenario 8	Flujo Básico	Flujo alterno 3: presionar cancelar		

Fuente: Paso 1: Identificar escenarios. Extraída de Méndez, Pérez y Mendoza (2007)



Escenario	ID caso de prueba	Nombre del caso de prueba	Resultados esperando	Nivel de prueba	Tipo de prueba
2	1	Login con caracteres inválidos	Mensaje Error 20-login inválido	Sistema/ Acceptation	Función/ Seguridad
2	2	Login con caracteres minúsculas	Mensaje Error 20-login inválido	Sistema/ Aceptación	Función/ Seguridad
2	3	Login con longitud mayor a 10 caracteres	Mensaje Error 20-login inválido	Sistema/ Aceptación	Función/ Seguridad
2	4	Login en blanco	Mensaje Error 20-login inválido	Sistema/ Aceptación	Función/ Seguridad
2	N				

Fuente: Paso 2: Identificar CP. Extraída de Méndez, Pérez y Mendoza (2007)

3

Especificación CP

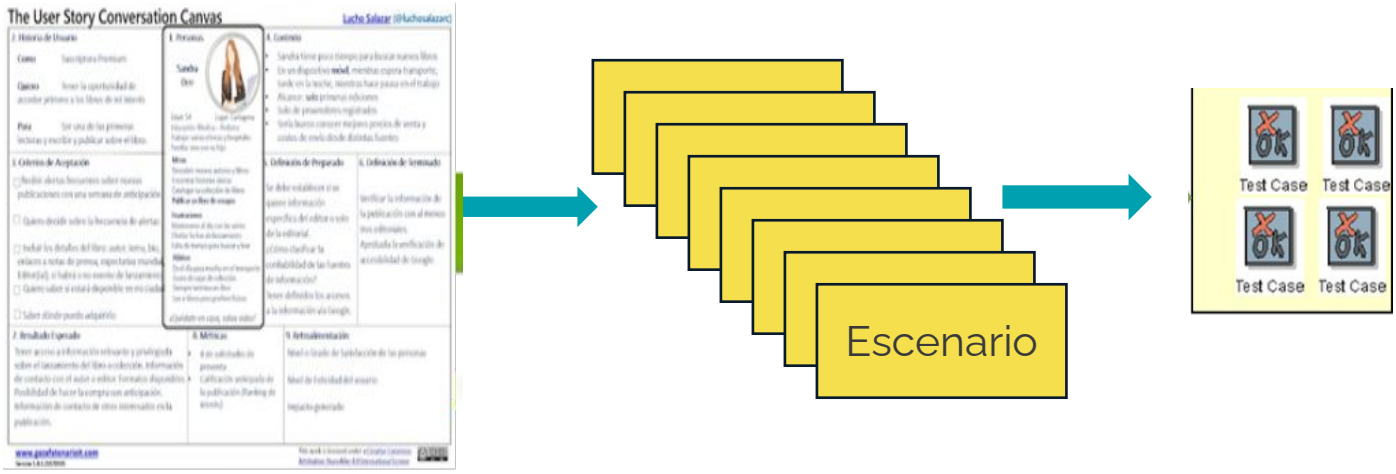
9 pasos

Diseñador de Pruebas

ID/Nombre sistema/ proyecto		Nivel de prueba	
ID caso de uso		Tipo(s) de prueba(s)	
ID requerimiento		Autor del caso de prueba	
ID/Nombre de escenario		Nombre del probador	
ID/Nombre caso de prueba		Fecha de Creación:	Fecha de ejecución:
Condiciones para que se ejecute el caso de prueba:			
Para la ejecución del ID caso de prueba:			
Nro. paso	Condición	Valor(es)	Resultado esperado	Resultado obtenido
Criterios de aprobación del caso de prueba:				
Decisión de aprobación del caso de prueba: Aprobó:____ Fallo:____				
Fecha de aprobación del caso de prueba:_____				

Fuente: Paso 3: Especificar los CP. Extraída de Méndez, Pérez y Mendoza (2007)

Ahora bien, si estamos siguiendo un enfoque ágil, para el cual elaboramos historias de usuario, podemos hacer la siguiente variante de MECAP:



Fuente: Variante de MECAP. Extraída de Méndez, Pérez y Mendoza (2007)

Recuerda que a la plantilla donde especificamos la historia, por regla general, se le agrega un **Diagrama** de actividades. De ese diagrama obtendremos los escenarios en este enfoque. De aquí en adelante continuamos con los pasos de MECAP antes descritos.

Este método nos garantiza la trazabilidad y que seamos exhaustivos a la hora de formular casos de prueba funcionales

CIERRE

La literatura es muy confusa cuando se trata de hablar de pruebas de software. Con este tema tratamos de darle un poco de **orden**, precisando las dimensiones de la prueba, las cuales se deben definir para poder formular un enfoque de prueba. Este enfoque es el "cómo vamos a probar". Este, a su vez, debe estar incluido en un plan de prueba que describa la estrategia de prueba.

Si una organización quiere que su software (app) sea el elemento diferenciador, debe hacerle una verificación continua de la **calidad**, la cual implica asignar recursos y establecerlos en un plan de acción. Para hacer más eficiente la verificación de la calidad se sugiere el uso de herramientas y métodos durante las actividades de prueba. Aquí se describió un método para formular casos de prueba funcionales, ya sea bajo el enfoque clásico o el ágil.

BERMEJO BARRERA, José C. (1991): *Fundamentación lógica de la historia*, Madrid, Akal.

BLOCH, Marc (1982): *Introducción a la historia*, México, Décima reimpresión, Fondo de Cultura Económica.

CARRERA DAMAS, Germán (1994): *Aviso a los historiadores críticos*, Caracas, Ediciones Ge.

CARRERA DAMAS, Germán (1981): *Metodología y estudio de la historia*, 2da. edición, Caracas, Monte Ávila Editores.

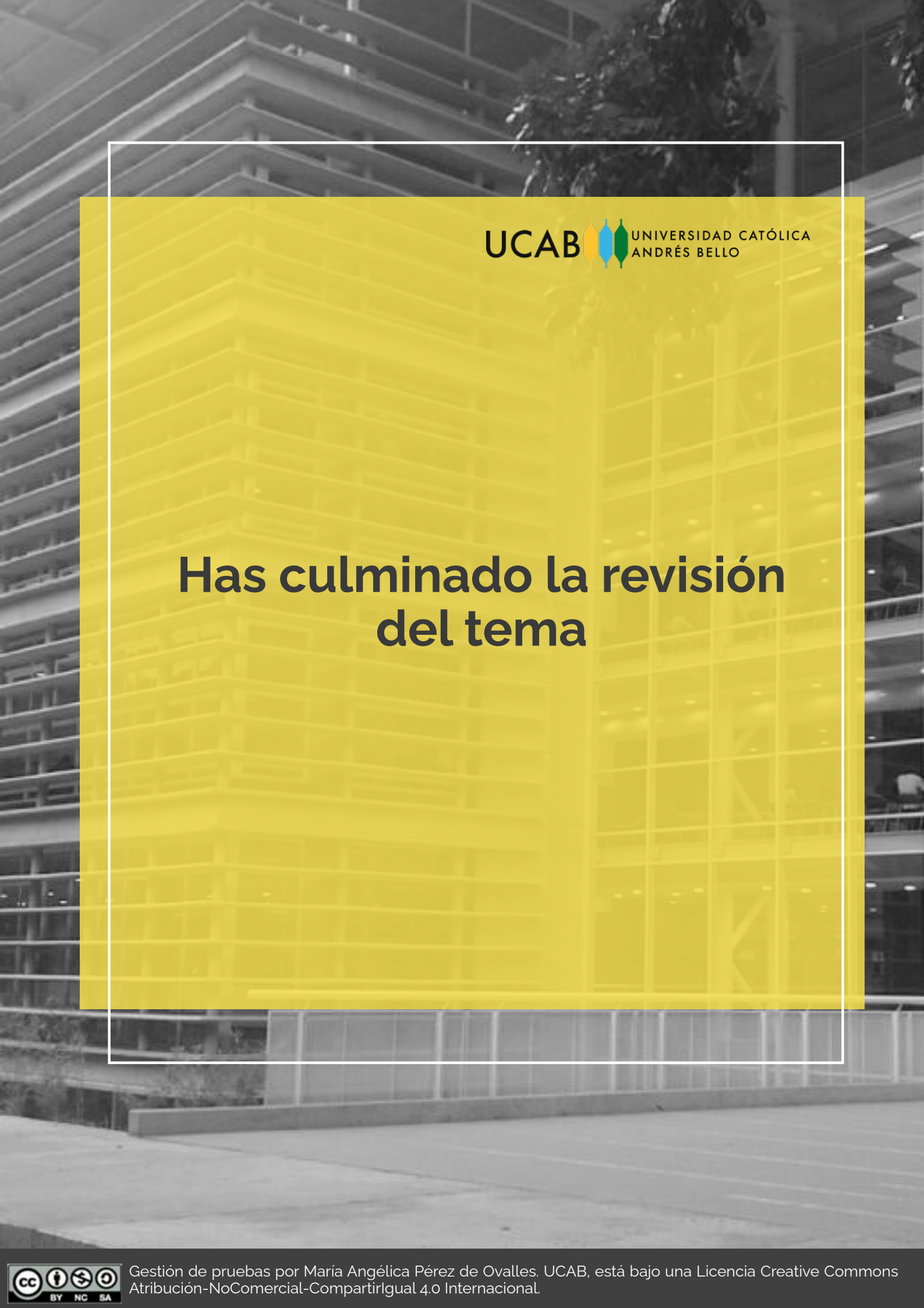
COLLINGWOOD, R.G. (1952): *Idea de la historia*, México, Fondo de Cultura Económica.

GRAFTON, Anthony (1998): *Los orígenes trágicos de la erudición. Breve tratado sobre la nota al pie de página*, México, Fondo de Cultura Económica.

GRAFTON, Anthony (2007): *What was History? The Art of History in Early Modern Europe*, Cambridge, Cambridge Press University.

LANGLOIS, Charles-Victor; y Charles Seignobos (2003 [1898]), *Introducción a los estudios históricos*, Alicante, Universidad de Alicante.

TENORIO-TRILLO, Mauricio (2019): *Clio´s Law. On history and language*, University of Texas Press.



UCAB UNIVERSIDAD CATÓLICA ANDRÉS BELLO

Has culminado la revisión del tema

Gestión de pruebas por María Angélica Pérez de Ovalles. UCAB, está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.