

Módulo 1 / Encuentro 2/17

Ciclo de desarrollo de software 1/2

OBJETIVOS DEL MÓDULO 1

¿Qué habilidades desarrollarás?

- Aprendizaje cooperativo entre pares
- Atención al detalle
- Fundamentos de la lógica de programación
- Manejo y priorización de la información
- Herramientas mínimas de seguridad de la información

¿Qué herramientas técnicas aprenderás?

- Entendimiento del mundo del testing
- Ciclo de desarrollo de software
- Introducción al desarrollo ágil
- Lenguaje unificado de modelado (UML)
- Terminología fundamental

Introducción y rompehielo



¡Te damos la bienvenida a tu segundo encuentro de trabajo!

Esperamos que hayas tenido un excelente equipo en tu primera sesión. Hoy conocerás a un equipo nuevo. Recuerda reconocer con un pulso a aquellos integrantes que hoy colaboren más con tu aprendizaje. Y si valoras al equipo completo, ¡no dudes en entregarle un pulso al equipo completo! Así te estarás asegurando que nuestro algoritmo te asigne en los próximos encuentros con personas que colaboren con tu aprendizaje.

¡Demos comienzo a la actividad del día de hoy!

Presentación del equipo:



No dejes de hacer la pequeña ceremonia de presentación. Toma tan solo unos minutos y cambia la experiencia de todo el equipo. Indica tu nombre y de dónde vienes. Ya sabes: puedes hacerlo en el chat si no deseas romper el hielo tú primero.

Les dejamos una pregunta para abrir la sesión (si lo desean):

¿Tienen alguna experiencia con programación? ¿Saben por qué es relevante saber algo de programación para dedicarse al testing?

Utilicen unos 10 minutos para compartir estas breves presentaciones. ¡Anímate! Quienes están contigo en el equipo de hoy son parte de la gran comunidad que está aprendiendo junto a tí.

¡MANOS A LA OBRA!



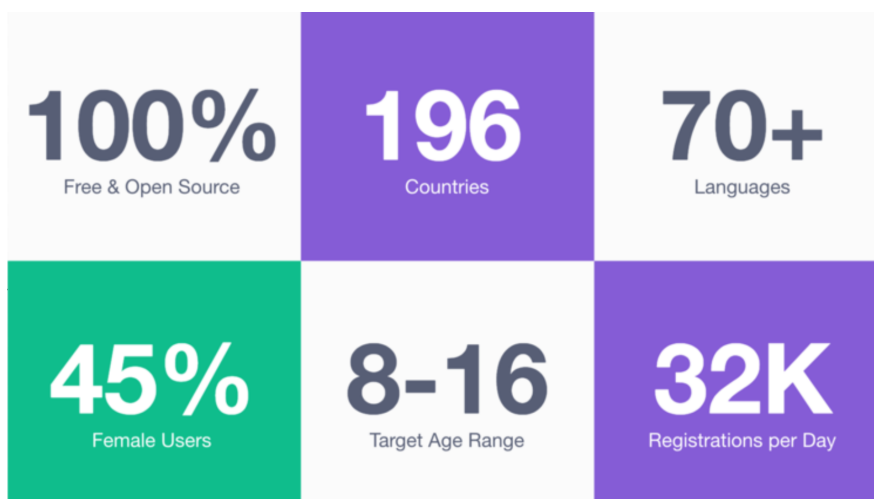
SCRATCH

Hoy vamos a tomar un pequeño desvío por el mundo de la creación de software. Para ello, estaremos utilizando **Scratch** que será necesario para resolver las actividades del encuentro de hoy.



Secreto de la industria: Scratch fue creado por el MIT Media Lab, uno de los centros de innovación tecnológica más prestigiosos del mundo. Lleva 15 años cumpliendo su misión de que niños y jóvenes de todo el mundo den sus primeros pasos en el mundo de la programación. Scratch es completamente gratuito, es usado por millones de personas en todo el mundo y es engañosamente simple.

Parece un juego para niños, pero varios de sus desafíos son difíciles de resolver hasta para programadores senior, ya que el desafío está en la lógica de cómo pensar el programa y no tanto en la resolución ya que Scratch funciona a base de imágenes y no código de texto.



Tendrás que realizar 3 actividades que serán cada vez más complejas.

¿Las realizo en equipo o en forma individual?

Esto es lo que nosotros sugerimos: que comiences en forma individual para poder detectar esas partes que te salen más fácil y esas partes que te cuestan más.

Pueden acordar como equipo destinar, por ejemplo, unos 20 min a entender la ejercitación e intentar resolverla en forma individual y luego dedicar unos 25 minutos a resolver punto por punto en equipo, apoyándose en las fortalezas que traiga cada integrante del equipo de hoy.



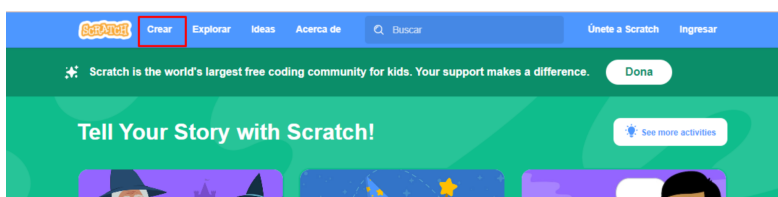
¿Te interesa ahondar un poco más en los conceptos de la programación? Aquí te dejamos un **Anexo** con algunas definiciones. Recuerda que este es **material extra**, su lectura no está contemplada en el desarrollo de la jornada.

[W QA E2 - Bases de la programacion.docx](#)

Ejercicio #1

- Accede a Scratch - <https://scratch.mit.edu/>

Nota: no es necesario crearte un usuario, puedes ir directo a la sección "CREAR"



- **Descripción del desafío general:** Lograr que el personaje realice una serie de acciones que le indicamos. Para ello debemos generar una lista de comandos para que el personaje realice acciones precisas.

A- Que se desplace de izquierda a derecha unos 30 pasos y pregunte tu nombre.

B- Que repita la acción anterior 6 veces y se detenga por si solo.

¿Lo has logrado? No dejes de ver los tutoriales de Scratch, ya que son breves y muy claros.

¿Para qué estamos haciendo este ejercicio? Lo que estamos resolviendo son algoritmos muy básicos. Esto nos permite comprender cómo está hecho el código en un programa o software. Y luego como testers tendremos la posibilidad de identificar mejor en dónde es que esta serie de instrucciones está fallando. Podremos ser más precisos y efectivos, entregando informes muy valiosos para mejorar el producto para el que estemos trabajando.

Un algoritmo es una serie de guías que describen cómo realizar una tarea. Piensa en un algoritmo como una serie de instrucciones paso a paso que crean un patrón predecible en una serie de números o unas líneas de código.

Ejercicio #2

A- Lograr que el gatito dé 3 vueltas alrededor de la pantalla. ¿No sabes cómo debería lucir? Aquí tienes un [link](#) a una muestra del resultado final esperado.

B- Accede a este [link](#). Analiza el código y realiza los siguientes cambios:

- b.1. Cambia el gatito por un personaje que te represente a ti.
- b.2. Cambia el obstáculo por uno más realista.
- b.3. Cambia el texto de lo que dice el personaje por un texto que desees o consideres apropiado para compartir con el equipo de hoy.

Ejercicio #3 - Desafío oficial de Scratch

<https://scratch.mit.edu/projects/114805446/>

¿Te animas a replicar lo que ves en pantalla?

Tienes ***dos caminos posibles***:

- **Camino A**, para quienes ya tienen algo de experiencia en escribir código

- **DISEÑO:** Escribir lo que te parece que debes realizar, armar un pequeño diagrama de los pasos a realizar. Documentar ese diagrama (no solamente hacerlo en forma mental).
- **EJECUCIÓN:** Correr el programa y cotejar contra el ejemplo si ha fallado algo en el diseño inicial. Documentar los cambios que realices para ajustar.
- **DOCUMENTACIÓN FINAL:** Escribe el flujo en un diagrama y luego muestra en pantalla cómo funciona tu versión. Puedes hacer mejoras estéticas, de personajes, pero no funcionales.

- Camino B, para quienes hoy están probando escribir código por primera vez

- **DOCUMENTACIÓN:** Imagina que debes solicitar a alguien que realice un programa que cumpla con los requisitos que ves en pantalla. Piensa en todas las funciones y escríbelas en este documento.
- **CAMBIOS:** Entra a inspeccionar el código y realiza cambios. Observa cómo estos cambios afectan el comportamiento del personaje - ¡o cambia los personajes para cambiar la historia!
- **EJECUCIÓN:** Intenta resolver parte del desafío en un proyecto iniciado por tí. Fíjate hasta dónde llegas y qué te falta entender para poder lograr el desafío completo.

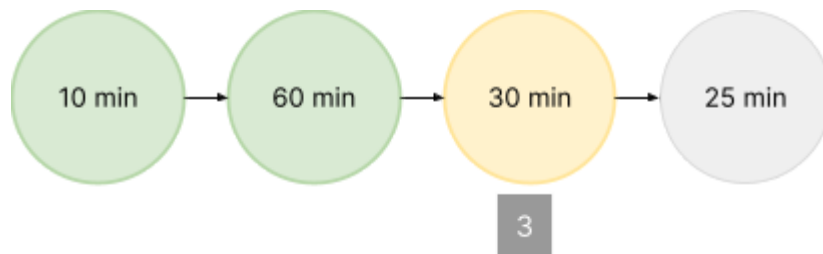
¡Hora de volver al equipo!

Hemos trabajado mucho en el encuentro de hoy... y todavía nos queda un poco más.



Júntense nuevamente y conversen sobre los ejercicios realizados. Enfoquen sus preguntas en ver cómo resolvió cada miembro del equipo los requerimientos de cada desafío. No existe un solo camino para resolver un pedido, ¿o sí?

Compartan pantalla, analicen en dónde hicieron las cosas distinto.



ATENCIÓN: antes de seguir, hagan un check de tiempo. ¿Cómo vienen?
¿Necesitan apurarse? ¿O resultaron ser unos genios de la organización?

Retomamos el Ciclo de vida de la Producción de software que vimos en el encuentro pasado y que hoy vivimos en primera persona con el ejercicio de Scratch.

¿Han podido identificar alguna de estas etapas en los ejercicios que realizaron en Scratch?

Si bien no estaban resolviendo los requerimientos de un cliente, de alguna manera intentaron descubrir qué se necesitaba resolver y cómo lo iban a hacer.

Nota: ¿Realizaron alguna secuencia de testing en la resolución de los ejercicios anteriores?

Ciclo de producción de software: *testing*

Cuando el código es pequeño y no es complejo es relativamente sencillo hacer un pequeño test para comprobar que esté funcionando. Cuando somos desarrolladores a eso lo llamamos *unit testing*.

Si durante la resolución de los desafíos en Scratch, realizaron pruebas intermedias para comprobar que el código estuviera actuando como ustedes lo habían imaginado, estuvieron realizando acciones de *unit testing*.

En programación, una *prueba unitaria* o *test unitario* es una forma efectiva de comprobar el correcto funcionamiento de las unidades individuales más pequeñas de los programas informáticos. Por ejemplo, una función o un procedimiento.

En el ciclo de vida de producción de software, ese tipo de testing se realiza durante la etapa de *development* o desarrollo y la realizan los equipos de desarrolladores.

Etapas de testing:

Una vez que el código ha sido chequeado por parte del equipo de desarrollo, se envía al equipo de Quality control (QC) para que revisen si funciona. Aquí nos adentramos en el corazón del trabajo de un equipo de testing.

¿Cómo se prueba el software?

Se realiza un empaquetado para poder implementar el software en un ambiente especial de pruebas para garantizar la calidad. Las pruebas o el control de calidad garantizan que las soluciones implementadas superen el estándar de calidad y rendimiento. Esto puede implicar repetir pruebas unitarias, la realización de pruebas de integración y de extremo a extremo, verificación/validación e informes o identificación de errores o defectos en la solución de software.

Aquí el tester es protagonista, debe encontrar los fallos cometidos en la etapa o iteración y reportarlos, para resolverlos antes de la siguiente etapa o iteración. Su éxito se medirá en su capacidad de encontrar estos errores antes que el usuario.

Pro tip: El momento del testing es fundamental.

¿No nos crees? [Lee en este enlace](#) los costos de los errores más caros de la historia por falta de pruebas exhaustivas.

¿Cuándo comienzan las pruebas de software en SDLC?

La mayoría de la gente piensa en las pruebas de software como algo que sucede al final del proceso de desarrollo de software. Después de todo, ¿qué podría ser más lógico que probar el producto que se acaba de crear?

En realidad, las pruebas de software deben comenzar mucho antes de que el producto terminado esté listo para enviarse. De hecho, a menudo es mejor comenzar a probar temprano en el SDLC, cuando todavía hay muchos cambios por hacer.

La primera fase del SDLC (**strategy**) incluye la **recopilación o análisis de requisitos**.

Aquí es cuando determina qué debe hacer el sistema y cómo funcionará. También es cuando se identifican los posibles riesgos y problemas que deben abordarse.

Como todavía no hay nada del producto desarrollado, es como ver los planos de una casa a punto de ser construida. Podemos anticipar algunos problemas antes de gastar dinero en construir, pero otros solamente los podremos ver una vez que se inicia el desarrollo del código.

El diseño de las pruebas debe comenzar durante la recopilación de requisitos para garantizar que el sistema funcione según lo previsto y que no haya fallas importantes. Si se encuentran problemas en esta etapa inicial, a menudo se pueden solucionar antes de que se invierta demasiado tiempo y dinero en el desarrollo.¹

¹ No olvidemos que las metodologías ágiles de desarrollo nos permiten entregar *MPVs* (o productos mínimos viables, por sus siglas en inglés). Estos MVPs muchas veces son tan solo una maqueta del sistema que estamos desarrollando y ya nos permiten poder evaluar si lo que estamos construyendo avanza en la dirección correcta o no.

Una vez que se finaliza la documentación de los requisitos, se procede al diseño (*design*) y desarrollo (*development*). Aquí es donde tiene lugar la escritura de código real y el producto comienza a tomar forma. El código debe probarse a medida que se escribe, para asegurarse de que funciona correctamente y cumpla con todos los requisitos.

En este punto, las pruebas no solo se preocupan por la funcionalidad; también comienzan a abordar el rendimiento y la escalabilidad.

El producto debe poder manejar la carga esperada y ofrecer tiempos de respuesta aceptables. *(¿Te suena familiar que la página web de un concierto se "caiga" en las primeras horas de venta de entradas? Eso responde a la carga que puede sostener un proyecto de software)*

A medida que avanza el desarrollo, se realizan pruebas más detalladas sobre características y funciones específicas. Las pruebas de integración se realizan para asegurarse de que todas las piezas encajan correctamente, y las pruebas del sistema verifican que todo el producto funciona según lo previsto.

Todo el conocimiento que se gane en la etapa de análisis de requerimientos, permite que los QA, o los testers si es un equipo pequeño, puedan ya ir trabajando sobre el diseño de las pruebas. No es necesario esperar a que los desarrolladores entreguen porciones de software para comenzar a trabajar. Y muchas veces las preguntas de los testers ayudan a construir un código más eficiente y funcional ya que ponen foco en problemas que pueden ocurrir más adelante y permiten que se solucionen desde el momento del diseño.

MATERIAL DE LECTURA

Introducción al lenguaje unificado de modelado (UML)

¡Pro tip alert! Esta introducción a UML es tan solo eso. Para poder utilizarlo correctamente se necesita ir más profundo y estudiar. Si es de tu interés, no dudes en hacer una rápida búsqueda en Google para aprender más sobre cómo y cuándo usarlo.

Contexto

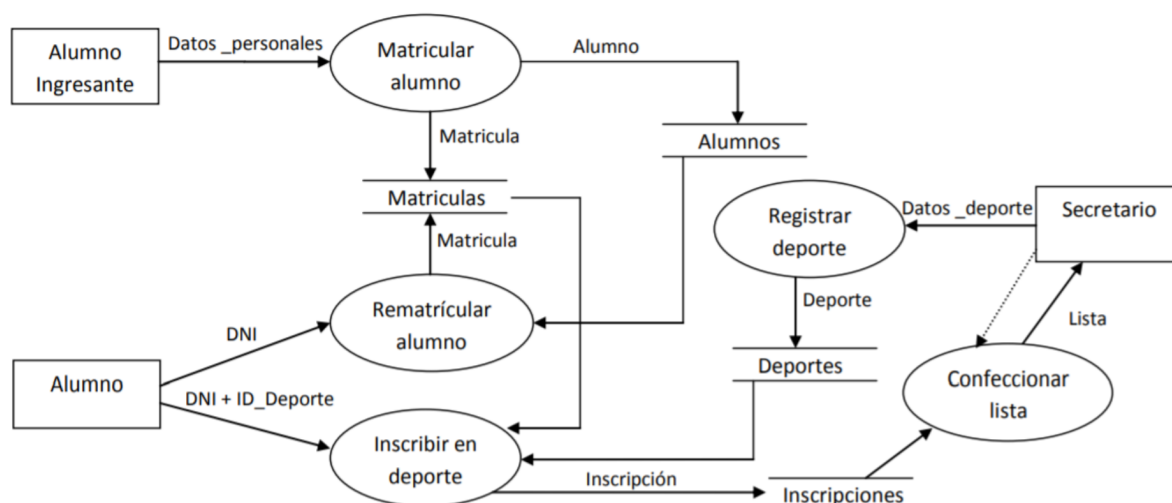
Uno de los mayores desafíos en el diseño y producción de software es poder entenderse entre equipos de distintas disciplinas. El cliente pide requerimientos en lenguaje cotidiano, los ingenieros lo pasan a lenguaje técnico y los testers deben poder entender ambos lenguajes para poder controlar que lo que se pidió esté realmente presente en el producto o proyecto que se va a entregar.

Una de las estrategias que se utilizan para diagramar los requerimientos es el uso de esquemas realizados en UML o *unified modelling language* - lenguaje unificado de modelado. Cuenta la leyenda que en el año 1997 un grupo de ingenieros entusiastas, cansados de intentar leer dibujos en servilletas, decidieron poner fin al castigo y armaron este sistema de símbolos y códigos que es *independiente de los lenguajes de programación*. ¿Qué significa eso? Que no importa el nivel de requerimientos o la dificultad del lenguaje a utilizar, las funcionalidades y casos de uso se pueden representar utilizando UML.

Como UML es un lenguaje y no una metodología, se puede utilizar sin necesidad de tener guías o ceremonias.²





¿NECESITAS UN EJEMPLO?

El director de un Centro de educación Física, los ha contratado para desarrollar un sistema, pero solo cuenta con una gráfica del diseño preliminar:



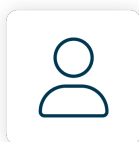
² En el caso de las metodologías ágiles se llaman así porque son metodologías y requieren que aprendamos ceremonias, pasos y procesos para llevarlas a cabo correctamente y que nos brinden los mayores beneficios al ser implementadas. Con solamente *saber* que las metodologías ágiles *existen* no alcanza.

Este diseño utiliza UML para explicar las relaciones entre los datos, los roles, las transacciones y la dirección de flujo de datos. A continuación tienen una ayuda memoria para poder analizar en profundidad el diagrama.

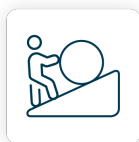
Símbolo	Descripción
	Las dos líneas indican una tabla de datos (de una base de datos), como ser Matrículas, Alumnos, etc.
	Los rectángulos son los roles, tales como secretario, alumno, etc.
	Los óvalos son acciones o transacciones que se pretenden llevar a cabo.
	Las flechas indican flujos de datos, con la información entrante o saliente.

¡MANOS A LA OBRA!

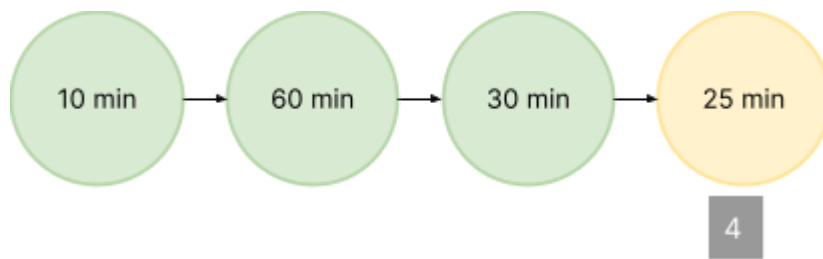
Ejercicio #4



1. Analiza en forma individual el gráfico de requerimientos. Considera el cuadro con el ayuda-memoria de UML.
2. Describe los requerimientos del sistema en formato de lista completa y siguiendo las secuencias completas. Controla que no falte ningún dato ni acciones.



Este es un ejercicio que requiere gran atención al detalle. Exige que puedas discriminar las acciones y el orden en que ocurren. Es importante que lo realices en forma individual primero, antes de recurrir a tu equipo. Ese esfuerzo que hagas hoy, tendrá su recompensa más adelante: verás que te es más fácil consolidar los conocimientos. Salir de la zona de confort no es fácil.



¡Hora de cerrar!

Consolidación de cierre:



Tómense estos últimos 25 minutos del encuentro de hoy para comparar los resultados en el ejercicio que realizaron recién.

- Uno de ustedes puede compartir pantalla mostrando su resolución y el resto puede aportar:

1. Lo que hicieron igual
2. Lo que resolvieron distinto
3. Evaluar si alguna versión es superadora y tomar nota de por qué. ¿Qué elementos de esa versión nos llaman la atención por su claridad? ¿Será la forma de comunicarlos?
4. ¿Algún integrante del equipo de hoy tenía ya experiencia con este lenguaje y puede hacer aportes que le sirvan al resto?

¡Hora de regresar a la sala general!

No olvides agradecer a tus compañer@s de hoy. Recuerda sus nombres ya que posiblemente los vuelvas a ver pronto. Y si deseas valorar el aporte de alguien en particular, ¡entrégale un pulso!