Manual

Tabla de contenidos

Introducción

¿Qué es el software?

Preguntando esto a Google (que es un software) nos dice:

- Software: Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.
- Origen: Préstamo del inglés *software*, compuesto por *soft* 'blando' y *ware* 'utensilios, objetos'. Término creado por los ingenieros informáticos americanos por analogía a *hardware*.
- Como hace referencia a la palabra "hardware", entonces la buscamos para ver qué es:
- Hardware: Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático.
- Origen: Préstamo del inglés *hardware*, compuesto por *hard* 'duro' y *ware*, elemento que entra en la formación de muchos términos ingleses con el valor de 'utensilio, artículos'. El término fue creado en los años cincuenta por ingenieros informáticos junto a *software*

Ejemplos

- Software: *Facebook, aplicaciones de celular, chat, Windows, Google, la página del diario El País, el reloj de la computadora, la calculadora, etc.*
- Hardware: *mouse (ratón), teclado, monitor (pantalla), tarjeta de red, disco duro, etc.*

¿Para qué sirve el software?

- El software es para hacer algo en particular.
- Todo software tiene un fin. En base a una necesidad surge el software.
- Está en todos lados.
- Es lo que uno no puede tocar de una compu. En contraposición, el hardware es todo lo que uno puede tocar.

¿Qué es el Testing?

Primero, pensemos en ¿qué es la calidad? La calidad es la satisfacción del cliente hacia un producto o un servicio. No tenemos que alcanzar la perfección, ya que no existe la perfección, no existe un software perfecto, sino tratar de tener un producto lo más adecuado posible al cliente, al usuario, y así poder lograr su satisfacción.

Cem Kaner define el testing como una investigación técnica de un software con el fin de brindar información sobre la calidad de la misma, a las diferentes personas involucradas en su construcción.

Con esta información se pueden tomar decisiones: por ejemplo cómo saber cuándo liberar un producto (para que la gente lo empiece a usar, lo pueda comprar, etc.), o cómo mejorar las diferentes áreas dentro de la empresa.

Uno con esta actividad puede trabajar la confianza, dar información de qué tanto podemos confiar en el software. La confianza es fundamental, ya que, si me instalo una aplicación y no me da confianza, entonces no la querré usar más. Si me instalo un juego en el celular, y no funciona, no lo usaré porque no me divertirá. Esto implica que la empresa que desarrolló ese juego no obtenga ganancias, no se haga popular, no triunfe. ¡El tester ayuda a prevenir estos problemas!

Ejemplos de “testing” cotidiano

- Alguien te pasa su número de teléfono, y después de guardarlo en tu celu, llamas para probar que anotaste bien o se lo mostrás para verificar, etc.
- Sacas una bebida de la heladera del súper, y la tocas para ver si está fría.
- Compro un café, toco para ver la temperatura, lo huelo para ver si estará bueno, lo pruebo. Si me gusta, considero que tiene buena calidad.

¿Qué es un tester?

Es un oficio (como el carpintero, panadero) pero en la industria del software.

- Es quien se encarga de verificar si una aplicación funciona como debe. Si cumple con la necesidad para la que fue creada.
- Que se cumpla todo lo que está especificado en el documento que explica cómo debería funcionar el software (lo veremos más adelante).

Un tester es un investigador. Básicamente se dedica a investigar si un software es de calidad o presenta errores o problemas, siempre pensando en las personas que lo utilizarán.

Para ser un buen tester es necesario desarrollar un conjunto amplio de habilidades, como por ejemplo la comunicación, trabajo en equipo, pensamiento crítico, poder de observación, ser meticuloso y detallista, tener conocimiento del negocio (si es una aplicación de ventas, ¡conocer de ventas!), de la aplicación que se prueba y de cómo planificar, diseñar, ejecutar las pruebas.

Ejemplos de tareas que hace el tester:

- ¿La aplicación funciona para MP3 y WAV?, ¿la lectora de cd?, ¿qué pasa si funciona con el WAV, qué pasa si no?
- Si la aplicación no cumple con la necesidad pedida, se debe informar al desarrollador. Veremos cuáles son los mecanismos para esto.

Requerimientos

Cuando un cliente solicita el desarrollo de un software está describiendo algo que todavía no existe y por lo tanto es importante que se pueda definir bien cómo debe funcionar ese software (que debe hacer, y que no debe hacer). No es lo mismo ir a comprar un teléfono Android Nexus 4.0 (que es un producto concreto que existe en el mercado) que contratar el desarrollo de una aplicación (por ejemplo, para manejar las compras online de un almacén).

A la definición de qué debe hacer y qué no es lo que se llaman los requerimientos de un software. Los mismos detallan cómo debe comportarse el software para satisfacer una necesidad de un usuario.

Los requerimientos pueden dividirse en requerimientos funcionales y requerimientos no funcionales.

Los **requerimientos funcionales** definen las funciones que el sistema será capaz de realizar – o **funcionalidades** – por ejemplo el manejador-de-compras online tiene entre sus funcionalidades "permitir buscar productos por nombre", otra de sus funcionalidades es de poder "comprar un producto seleccionado".

Los **requerimientos no funcionales** tienen que ver con características que de una u otra forma puedan limitar el sistema – como pueden ser requerimientos de performance, usabilidad, etc. Por ejemplo se podría requerir que la aplicación de compras del almacén no demorar más de 1 segundo en desplegar resultados de una búsqueda en el día previo a un feriado (cuando se espera la mayor cantidad de gente comprando), se podría también requerir que los usuarios pudieran seleccionar los productos con imágenes de forma que no necesiten leer los resultados de una búsqueda en una lista.

¿Cómo se definen los Requerimientos?

En un proceso de desarrollo de software vamos a encontrar a un cliente y un proveedor:

- **El Cliente** - que es quien tiene una necesidad y es quien solicita el desarrollo del software (generalmente también es el que paga por el desarrollo)
- El **equipo de proyecto** (donde estamos nosotros los testers) - que somos quienes vamos a participar en el proceso de desarrollo de ese software para satisfacer dicha necesidad.

Para que el cliente se anime a comprar (o a esperar por) algo que aún no existe es muy importante que el cliente y el equipo de proyecto se puedan poner de acuerdo sobre qué va a recibir el cliente cuando el proyecto termine, es decir, como va a funcionar esa aplicación – o cuales son los requerimientos del software.

Para ello en el equipo de proyecto vamos a encontrar alguien que cumple la función de "analista de negocios" que es quien se encarga de hablar con el cliente, capturar lo que el cliente necesita y definir los requerimientos.

Las formas más comunes en que el Analista define requerimientos son las siguientes:

- Especificación Funcional
- Historias de Usuario

Es importante que se logre la mayor claridad posible en cuanto a los requerimientos, debido a que en el desarrollo de software generalmente hay mucha gente involucrada por lo que se vuelve complicada y compleja la comunicación entre todos y asegurarse que frente a un requerimiento dado todos entienden la misma cosa.

Especificación Funcional de Requerimientos (ESRE)

El ESRE es un documento que detalla todos los requerimientos – o Requisitos – con que el software debe

cumplir.

Una buena especificación debe ser:

- Completa. Todos los requerimientos deben estar reflejados en ella.
- Inequívoca. La redacción debe ser clara de modo que no se pueda malinterpretar. Esto es digamos una utopía, porque cualquier redacción puede lugar a malinterpretaciones. Teniendo esto en cuenta es que se intenta incluir diagramas, prototipos y la mayor información posible para tratar de minimizar las malas interpretaciones.
- Priorizable. Los requerimientos deben poder organizarse jerárquicamente según su relevancia para el negocio y clasificándolos en esenciales, condicionales y opcionales.
- Modificable. Aunque todo requerimiento es modificable, se refiere a que debe ser fácilmente modificable.
- Trazable. Se refiere a la posibilidad de verificar la historia, ubicación o aplicación de un ítem a través de su identificación almacenada y documentada.

Para un requerimiento dado – por ejemplo el buscador de productos de una aplicación de compras online – una especificación de requerimientos detallaría algo semejante a lo siguiente:

"La pantalla de búsqueda tendrá un campo en la parte superior de la pantalla, donde el usuario podrá escribir el texto que está buscando – al lado habrá un botón que permitirá "buscar". Cuando el usuario hace clic en el botón de buscar, se desplegará debajo el resultado de la búsqueda que incluye todos los productos que contengan la palabra buscada en el nombre del producto o la descripción. El resultado de la búsqueda estará ordenado alfabéticamente e incluiría 20 elementos por cada página. La grilla de resultados se podrá navegar en forma standard – moverse entre las diferente páginas, incluir número de ítems en la lista, número de página en que el usuario se encuentra, etc."

Historias de Usuario

Otra técnica para definir requerimientos son las "Historias de Usuario". A diferencia del documento de Requerimientos, las historias de usuario intentan capturar suficiente información para que sea posible que desarrolladores y cliente tengan una conversación para entender que es lo que se intenta resolver. La historia de usuario va a capturar dado un requerimiento:

- A quién le importa
- Qué debe hacer en concreto
- Para qué.

Para la misma funcionalidad una "historia de usuario" diría lo siguiente:

"Como un comprador quiero poder buscar un producto por un texto contenido en su nombre o descripción de forma de no tener que recordar cómo se llama exactamente"

Como ven son dos formas muy diferentes de detallar cómo debe funcionar una aplicación – ambas detallan que el usuario que quiere comprar un producto en el sitio de carrito va a poder buscar productos por un texto, pero el nivel de detalles del resultado esperado es diferente en el ESRE (donde la idea es que este bien claro que y como debe funcionar el software) de la historia de usuario donde solo se explica el qué y por qué… y se deja que el equipo de proyecto defina el cómo junto con el cliente.

Equipos de Desarrollo de Software

Roles que participan

Un equipo de desarrollo puede ser una sola persona, o muchas, pero en cualquier equipo existen una serie de roles (funciones), que pueden ser identificados.

En un equipo pequeño, puede que una persona cubra múltiples roles, mientras que en equipos más grandes, es más común tener funciones dedicadas.

Independientemente del caso, la identificación de los roles en el equipo ayudará a estructurar el mismo, y a crear conciencia de las responsabilidades. Por ejemplo, si nadie se siente responsable de probar el software, será inevitable que se encuentren errores en la versión final.

El Cliente

Se puede pensar que tratar al cliente como parte del equipo de desarrollo es extraño, pero en realidad, no lo es: El cliente es un factor importante en el éxito de un proyecto, tanto como cualquier otro miembro del equipo, por eso es importante contar con la participación activa del cliente dentro del proyecto.

También es importante entender quién es en realidad “El Cliente”. Tanto si se desarrolla software para clientes actuales, como si se desarrolla para uno mismo, o para la propia empresa u organización, siempre hay un rol de cliente. El cliente, es en esencia, quien pone en marcha el proyecto, paga las cuentas, o define el resultado final. Aun si no se tiene literalmente un “cliente”, es bueno entender que aun así existe un rol “cliente” en su proyecto. Esto puede ayudar a evitar confusiones. Si hay varias personas diciendo que características se necesitan, hay que asegurarse de que exista algún responsable de tomar las decisiones cuando estos requisitos sean contradictorios.

El Analista

El analista es alguien que es responsable de entender las necesidades del cliente, y asegurarse de que la solución que está siendo desarrollada se ajusta a esas necesidades.

Las actividades típicas de un analista incluyen al relevamiento de requisitos, reuniones con clientes y la redacción de especificaciones funcionales.

Incluso si un proyecto es demasiado pequeño para escribir un verdadero documento de especificación, la comprensión de las necesidades del cliente es un trabajo importante, dado que a menudo el éxito de un proyecto de desarrollo depende de qué tan cerca está la solución desarrollada de las expectativas del cliente.

El Arquitecto

El papel del arquitecto es traducir los requisitos, tal como se define por el analista, en una solución técnica. Él

puede crear un diseño técnico, o simplemente algunos bocetos, de cómo el sistema va a estar estructurado. En cualquier caso, es la responsabilidad del arquitecto a pensar en el sistema antes de que se desarrolle. Si se hace bien, durante la fase de diseño se abordarán correctamente todos los problemas que se enfrenten en el desarrollo de la solución.

A menudo hay muchas maneras de lograr algo. El arquitecto de una aplicación es el que decide qué camino tomar, en base a la arquitectura global que ha elegido. Cuando el desarrollo se ha iniciado, es responsabilidad del arquitecto realizar un seguimiento del desarrollo, para ver si todavía se mantiene en consonancia con el diseño general.

El Desarrollador

El desarrollo efectivo de una aplicación es hecho por los desarrolladores del equipo. Pero un desarrollador tiene más responsabilidades que solo escribir código. Él es a menudo responsable de hacer el seguimiento de su propio progreso, e informar al jefe de proyecto de los problemas a los que se enfrenta. Él es también quien implementa las ideas del arquitecto, y como tal, puede tener que discutir las (in)posibilidades de la implementación con el arquitecto.

El Tester

Las pruebas son una parte importante para asegurar que el software funciona de la manera que debería. El papel de 'tester' se realiza a menudo por los desarrolladores para los aspectos técnicos y los usuarios para los aspectos funcionales. Un problema que surge de hacer a los desarrolladores probar su propio código es que, no importa lo bueno que sean, se ven influidos por la forma de su código fue creado. Cuando se prueba, se tendrá en cuenta esas mismas situaciones que que ya se tuvieron en cuenta a la hora de escribirlo. El tester es quien verifica la calidad del software que se está fabricando.

La calidad involucra muchos aspectos y es importante que el tester los tome en cuenta. Éstos son algunos:

- Asegurar que el software cumpla con las especificaciones del cliente.
- Asegurar que no haya errores introducidos por los desarrolladores.
- Asegurar que la aplicación o software fabricado sea consistente visualmente.
- Aportar un punto de vista de usuario para lograr una mejor experiencia en el uso de la aplicación.

El Líder de Proyecto

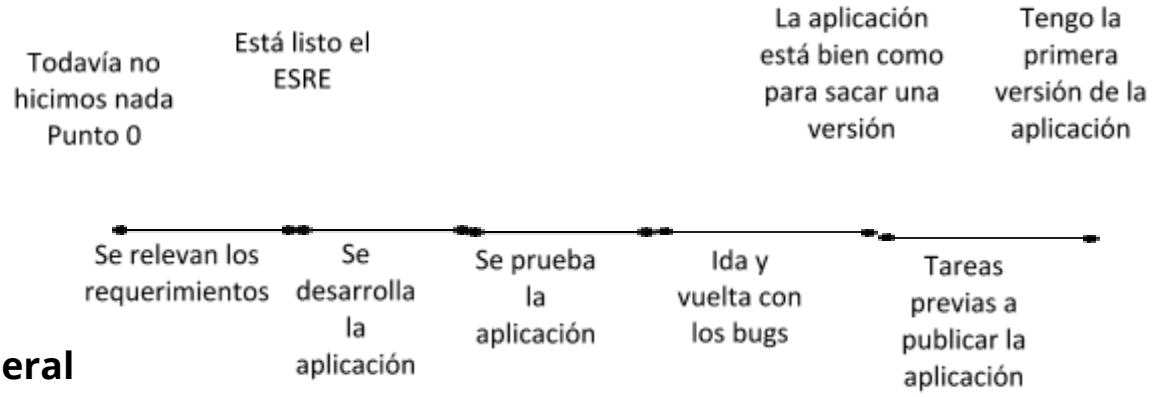
Un líder o gerente de proyecto tiene muchas responsabilidades. Es responsable de la planificación del proyecto, de mantener el proyecto dentro del presupuesto, y de la solución de problemas. En resumen, él resuelve cualquier problema que ponga en peligro el progreso del proyecto.

Muchas de las tareas del gerente del proyecto tienen que ver con la comunicación, la comunicación al cliente sobre el progreso del proyecto y la comunicación con todos los miembros del equipo. Incluso en los proyectos de desarrollo que no cuentan con un director de proyecto, es conveniente asignar el rol de gerente de proyecto a alguien, para que quede claro quién es responsable de la ejecución del mismo.

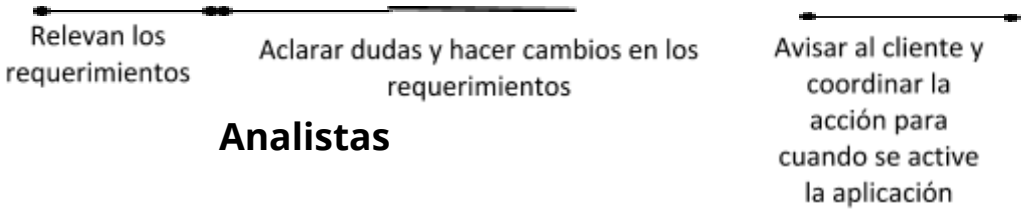
El Usuario

El usuario final es tanto o más importante que cualquiera de los otros roles. Algunas veces, el cliente y el usuario pueden ser la misma persona, pero puede pasar muchas veces que no lo sean. El usuario final es la persona que va a utilizar el software que se está fabricando. Si el usuario no queda satisfecho con lo que se hizo, es muy probable que se niegue a usarlo y así hacer fracasar todo el trabajo realizado. Hay que pensar como un usuario para intentar saber qué le gustará y qué no le gustará.

¿Qué hace cada uno en las distintas partes del proceso?

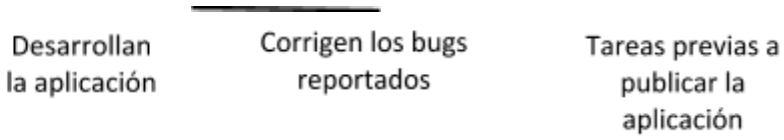


General

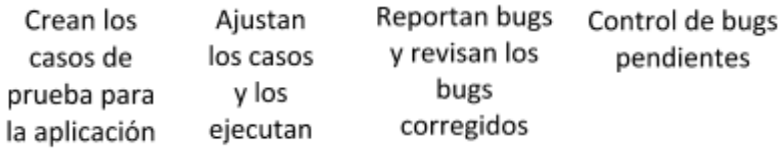


Analistas

Desarrollo



Testing



Aclaraciones:

Esto es una explicación simplificada de todo el proceso de desarrollo, desde que no hay nada hasta que sale la primer versión de una aplicación. Dependiendo del proceso, metodologías y detalle que apliquemos esto puede tener muchas variantes. Lo buscamos aquí es simplificar y que se entienda qué hacemos los testers y demás actores en cada momento durante proceso de desarrollo de la aplicación.

General: Es la línea de tiempo desde el punto 0, en el que no tengo nada hecho; o sea solo tengo la idea de hacer una aplicación. Hasta el punto, en el que saco la primera versión.

- Se relevan requerimientos: Aquí es cuando se intercambia información con el cliente para armar el ESRE, por lo que termina cuando el ESRE está pronto.
- Se desarrolla la aplicación: Aquí es cuando se hace todo lo necesario para tener una aplicación andando. Se programa el código, se ultiman los detalles de los requerimientos para decidir cómo se van a hacer, para la aplicación.
- Se hacen las pruebas: Acá entra el trabajo más fuerte de testing, se van haciendo las pruebas y levantando bugs.
- Ida y vuelta con bugs: Se, reportan, se corrigen, se cierran o pueden quedar para otra versión (no todos se van a corregir, sólo los que impidan que el usuario haga un buen uso de la aplicación).
- Tareas previas a publicar la aplicación: Que salga una aplicación, implica que todos en el proceso lleven a cabo ciertas tareas antes de que ésta se publique para los usuarios.

Analistas: Son quienes están en contacto con el cliente, y arman el ESRE basado en lo que pide. Pueden ser parte del equipo de desarrollo, o estar como un equipo aparte.

- Relevan los requerimientos: Ver lo de la general.
  Para los demás puntos no hay nada para agregar.

Desarrollo: Son quienes programan la aplicación (se puede considerar que los testers también están dentro de este equipo, pero tienen una tarea diferente por eso lo separamos).

- Desarrollan la aplicación: Ver lo de la general.
- Tareas previas a publicar la aplicación: son cambios técnicos para poder publicar, o sea todo lo que se necesita para que la aplicación quede disponible para los usuarios.

Testing: ¡Es nuestro rol!

- Crean los casos de prueba de la aplicación: O sea, los casos de prueba los arrancamos a crear cuando está listo el ESRE. No esperamos a que esté pronto algo de la aplicación para probar. Después los ajustaremos si vemos que se hizo como pide el ESRE pero de una forma diferente a la que esperábamos.
- Ajustan los casos y los ejecutan: Si a la hora de ejecutar ven que se hizo de una forma diferente a lo que pensaron pero cumple lo del ESRE, ajustan los casos. Ya van reportando los bugs que van encontrando.
- Reportan bugs y revisan los bugs que se corrigieron: Acá ya terminaron de ejecutar los casos, pero quedan bugs por corregir. Cuando un bug, que surgió de un caso de prueba se corrige se vuelve a correr el caso y los que puedan estar relacionados (capaz arreglaron una cosa, pero se rompió otra).
- Control de bugs: Como se dijo más arriba no todos los bugs se corrigen. Hay que ver si ya se corrigió todo lo más grave, y lo que queda; puede esperar a otra versión o hay algo que quiero corregir sí o sí, antes de publicar la aplicación.

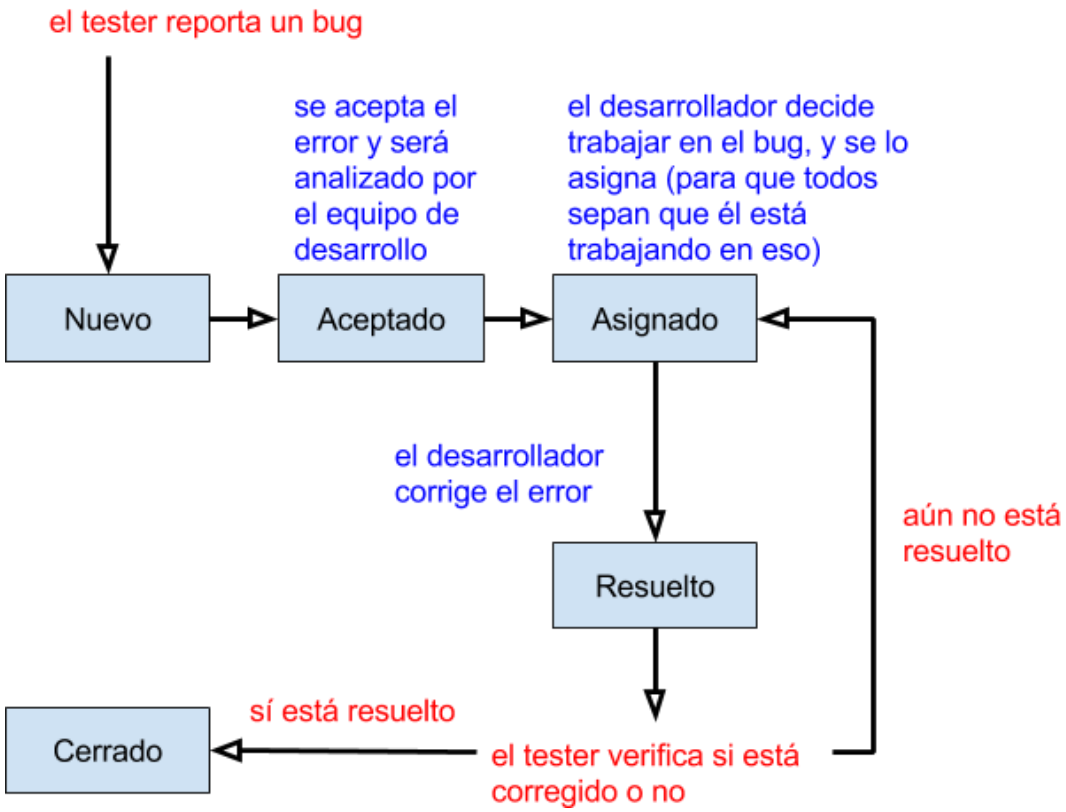
Mantis – Herramienta de Gestión de Bugs

Bug, bicho, incidente, defecto, error, son distintas formas de llamarle a lo mismo. Es lo que los testers estamos buscando todo el tiempo. Como una Mantis que busca bichos para comerlos, y de ahí el nombre de una de las herramientas gratuitas más famosas, Mantis Bug Tracker (gestor de bugs).

Ciclo de Vida de un Incidente

El flujo más típico de los incidentes (o bugs) es el que se representa en el siguiente esquema. Considera que en rojo aparecen las acciones que hace el tester y en azul aparecen las acciones que hace el desarrollador. Luego verás en las cajas las distintas situaciones en las que se encuentra un bug, las cuales son:

- **Nuevo**: un bug recién creado por un tester.
- **Asignado**: un bug que un desarrollador decidió corregir.
- **Resuelto**: un bug que un desarrollador corrigió y está esperando que un tester verifique que quedó correctamente corregido.
- **Cerrado**: un bug que ya fue resuelto y verificado por un tester, con lo cual ya no hay más para hacer.



Puede pasar que el desarrollador no entienda el incidente y necesite más información. Por eso también veremos que hay bugs que aparecerán como “Se necesita más información”.

Para evitar esto ver el repartido “Escribiendo Reportes de Error Efectivos”. Si llegase a suceder, el tester debe proveer más información y así el desarrollador podrá corregirlo.

También hay situaciones en las que un tester reporta un bug que quizá no era un bug, sino que el tester había entendido mal cómo debería funcionar el software, esperaba otra cosa, y por eso reportó el bug. En ese caso el desarrollador pasará el bug a “Resuelto” con una nota indicando que eso no era un error. El tester ahí deberá aceptar el comentario y cerrar el incidente, a menos que considere que sí es un error, con lo cual podría reabrirlo y continuar “discutiendo” con el desarrollador.

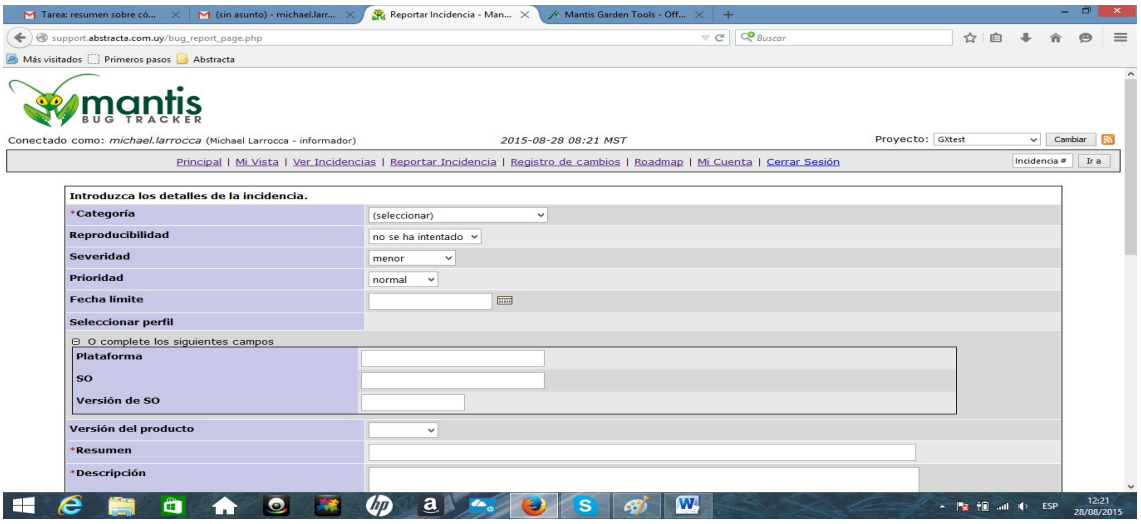
Realmente todos estos tipos de interacción serían muy complicados de seguir si no fuese porque contamos con herramientas que nos ayudan a ordenar nuestro trabajo.

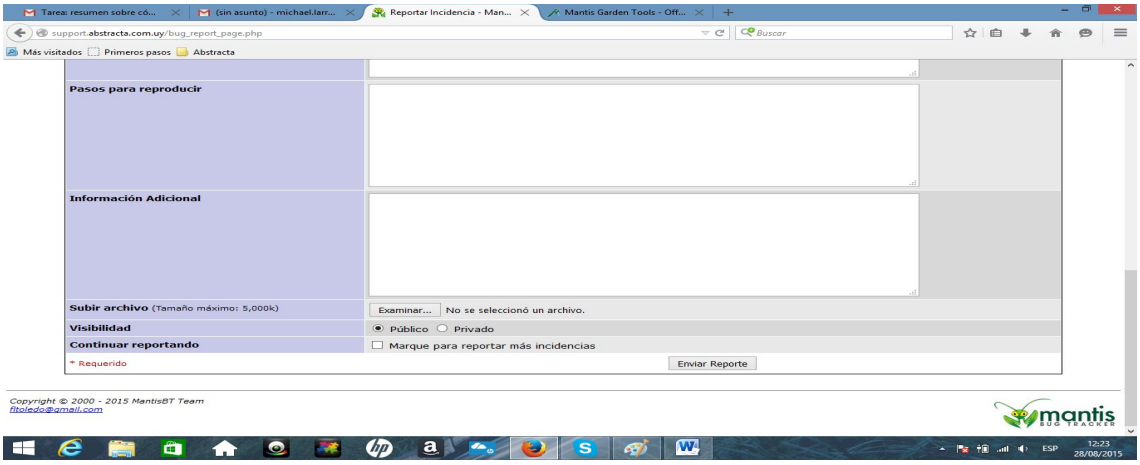
Uso de Mantis

Primero y fundamental es entrar a la página de mantis, después de haber ingresado a la página nos veremos en “Mi cuenta” que está marcado en una flecha verde, un poco más arriba hacia la derecha de “Mi cuenta” podremos observar que dice “Proyecto” (marcado con una flecha azul). Ahí seleccionamos el proyecto en el que queramos entrar, en mi caso yo ingresé al proyecto GXtest.

Una vez ingresado al proyecto para reportar incidentes, entramos a reportes de incidentes que como vemos en la imagen está en una flecha roja.

Una vez ingresado al reporte de incidentes nos veremos con una imagen en la pantalla que se las mostraré en 2 fotos.





Al encontrarnos con esa imagen veremos:

- Categoría
- Reproducibilidad: Esto es con la frecuencia que aparece el error al probarlo, ejemplo a veces, siempre, etc.
- Severidad: La severidad va de la mano con la prioridad, la severidad es el grado del incidente o sea mayor intermedio o bajo
- Prioridad: La prioridad es el grado depende de la persona que crea el diseño ejemplo: si tengo un celular y el que crea el celular me dice que le de enfoque a la falta de ortografía y si es que llega a tener falta de ortografía le pongo prioridad alta.
- Fecha Límite: Acá ponemos una fecha límite de entrega el reporte casi siempre se pide un día antes o 2, ejemplo si va a salir el Samsung galaxy s7 un viernes lo mejor sería que el reporte me lo envié un miércoles a más tardar jueves así tendremos tiempo para seguir reportando y a la hora de salir el producto esté en perfectas condiciones
- Versión del productor: la versión del producto es como el Windows a mediado que se va actualizando se va poniendo otra versión mejor ejemplo 7.0 / 7.5 / 7.5.1 / etc.
- Resumen: Acá en el resumen es como el título para indicar que es el tema por el cual vamos a reportar
- Descripción: La descripción a mi punto de vista es uno de los pasos fundamentales ya que hay que ser bien detallistas a la hora de poner la descripción, escribir paso por paso y si es necesario poner imágenes para el que lo reciba tenga bien claro los pasos a seguir así no hay confusiones y habrá un mejor dialogo y mejor trabajo en equipo.
- Pasos para reproducir: Este paso no es un paso muy necesario. En lo personal yo no pongo nada sino que trato de enfocarme mucho en la descripción ya que los pasos para reproducir es bien parecido a la descripción y si hacemos bien la descripción no es necesario poner los pasos a seguir.
- Información adicional: Informe adicional en lo personal tampoco creo que es muy necesario, esto sirve para poner algo extra a la descripción o los pasos para reproducir.
- Subir archivos/examinar: Esto sirve para poner alguna imagen para aclarar algún punto en especial.
- Visibilidad: pública/privada. Esto sirve para saber quién lo puede ver y quien no. O sea, si ponemos público lo podrán ver todos y, si ponemos privado, solo podrá verlo la persona a la cual se lo enviamos
- Continuar reportando: Esto sirve para cuando estamos apurados y tenemos muchos reportes para reportar ponemos ahí y al enviar aparece todo lo mismo para seguir reportando y te ahorras el tiempo de entrar a la página y volver a poner reportes de incidentes.

Cuando terminamos de hacer todos los pasos y verificamos lo que pusimos ponemos enviar, y listo, reporte enviado ☺

¿Qué NO puede faltar en un reporte de Error?

Los datos que todo reporte debería tener son:

- **Título:** que describa el error.
- **El nombre del tester:** En caso que haya preguntas de como reproducir el problema, es importante que los desarrolladores puedan identificar a la persona que reportó el incidente.
- **Fecha**: Es importante capturar la fecha en que el incidente se reportó para poder saber cuánto tiempo se demoró en resolver.
- **Aplicación y versión** Es importante detallar que aplicación estamos probando, y que versión de la misma se está probando.
- **Ambiente del Tester:** En caso de reportar un error web diríamos que navegador se estaba usando, en caso de ser un reporte de un incidente en un dispositivo móvil diríamos que teléfono/modelo/versión del sistema operativo.
- **Los pasos:** necesitamos comunicar cómo reproducir ese error, algo así como la receta para reproducir el problema. Es necesario detallar bien los pasos para poder reproducir el error y que no ocurra el caso del teléfono descompuesto.
- **Captura de Pantalla:** Al mostrar los pasos de como reproducir, o el resultado obtenido, ayuda mucho poder tener capturas de pantalla que le permitan al desarrollador ver el problema más claramente.
- **Resultado Obtenido:** El error de lo que obtuvimos en la prueba
- **Resultado Esperado:** Lo que deberíamos obtener según la especificación.
- **Importancia**: se suele tomar una escala de valores como *bajo, medio, alto, crítico,* … y con esto se registra la importancia que se le debería dar al error, en base a qué tanto impacto puede tener en el usuario si encuentra el error, y en base a la probabilidad que el error aparezca (es un error en algo muy usado, entonces habrá mayor posibilidad de que suceda).

Escribiendo reportes de error efectivos

Todo reporte de incidente es una comunicación escrita al equipo del proyecto sobre la calidad del software bajo prueba. Muchas veces es tu habilidad para comunicar los incidentes de una aplicación – y no la severidad inherente a los bugs – lo que determina que los incidentes sean corregidos o no.

Ustedes pueden estar pensando: “Pero… yo odio escribir y no soy bueno para eso! ¿Cómo puede depender el destino del incidente del texto del reporte?”

Es tentador pensar que los bugs hablan por sí mismos –que cualquier persona en su sano juicio puede automáticamente ver que cierto error es horrible y debería ser corregido. Pero lamentablemente, no es lo que sucede.

Pero la buena noticia es que su habilidad para comunicarse efectivamente con los desarrolladores y el equipo del proyecto, no está predeterminada por cómo les ha ido en la clase de español (o inglés) del liceo. No se trata de escribir una prosa fluida con palabras interesantes. Ni siquiera se trata de gramática y ortografía correcta. Se

trata de expresarse claramente, utilizando las palabras estrictamente necesarias para explicarse. Demasiadas palabras, y la idea se pierde entre las ramas. Muy pocas palabras y los demás llenan los espacios con sus propias suposiciones.

Se trata nada más que de saber exactamente qué es lo que hay que comunicar. Si no están seguros de la naturaleza del bug, entonces no importa cuán bien puedan escribir el reporte de error, igual nadie más sabrá tampoco de qué incidente se trata.

Este reporte de incidentes presenta cuatro cosas que ustedes pueden comenzar a hacer desde ya para aumentar las chances de que las personas presten atención a los errores que reportan.

Conozcan a su audiencia

En cualquier curso de escritura les dirán que deben conocer para quién están escribiendo. Los reportes de error no son una excepción. Hay al menos dos audiencias para cualquier reporte de error: la persona que tiene que corregir el error y la persona o grupo que decide el destino del bug (a veces ambas cosas pueden ser hechas por la misma persona, pero aun así en cada rol tiene intereses/necesidades diferentes).

Vamos a llamar “el desarrollador” a la primera audiencia – es decir la persona que debe corregir el error. El desarrollador – precisa pasos claros e inequívocos para reproducir el problema, necesita detalles precisos sobre lo que hicieron y vieron.

Llamaremos a la segunda audiencia el “Comité de Revisión de Errores” – es decir la persona o grupo que decide sobre el destino del bug. Ellos precisan entender las consecuencias de elegir no corregir un error. Para lograr que los errores se corrijan, su rol es ayudar al comité a ver que los riesgos de no corregir el error son mayores que los riesgos de corregirlo.

Cuanto más comprendan cómo trabajan sus Desarrolladores y el Comité, mejor pueden adecuar los reportes de error a sus necesidades. Hagan el esfuerzo de conocer personalmente a sus interlocutores. Si pueden participar de las reuniones del Comité de Revisión de Bugs, háganlo. Aprenderán mucho sobre cómo piensan los destinatarios de los reportes de error.

Elijan un buen título

La breve frase que describe el bug es generalmente denominada, el título o la descripción del error. Es la parte más importante del reporte de error. Los miembros del Comité de Revisión de Errores, a menudo deciden que un incidente puede aplazarse basándose solamente en el título; si éste es débil, los miembros podrían determinar que no vale la pena invertir más tiempo en el bug (al fin y al cabo, hay otros 145 más para revisar en las 2 horas siguientes.) He aquí algunos ejemplos:

Bueno

Se cae cuando sale por tiempo.

Demasiado largo

El programa se cae cuando elegís salir del menú de archivo, justo antes de que la base de datos se tornara inaccesible y estuvieras salvando los cambios en un registro.

Detalles insuficientes

El programa se cae.

Demasiado vago

Problema cuando la base de datos está fuera de línea.

El equipo del proyecto también usa el título para referirse al bug y para buscarlo. La gente tiende a recordar mejor las palabras que los números. Por lo tanto, en lugar de recordar el bug 23423 la gente retendrá el bug que “No se puede instalar en Windows 2000” y va a usar esas palabras claves para buscar el bug en el futuro.

Es difícil escribir una descripción de error buena y concisa. Es probable que pasen más tiempo elaborando el título perfecto para el bug que escribiendo el resto del reporte de error. Asegúrense que el título es suficientemente corto para ser desplegado enteramente en la pantalla del error (sin desplazamientos) y en los reportes generados por la herramienta de registro de incidentes. El título no tiene que ser gramaticalmente perfecto – tiene que ser corto e ir al grano.

Describe clara e inequívocamente los pasos

Estos pasos le suministran al Desarrollador la información sobre “dónde vive” el bug, para que pueda ser encontrado y corregido. También provee información al Comité de Revisión de Errores sobre las circunstancias en las que el error ocurre.

Bueno...:

1. Ejecutar la aplicación / Mantenimiento de Clientes
2. Editar un cliente
3. Modificar datos, pero no grabar todavía
4. Bajar el servidor de base de datos
5. Intentar grabar el cambio del cliente
6. Recibir un error de “timeout”
7. Cerrar la aplicación.

Resultado: el programa se cae.

Demasiado impreciso

(Demasiado margen para interpretaciones erróneas).

Bajar el servidor de base de datos, grabar, luego salir. Revienta.

Demasiada información extraña: se pierde lo que es relevante para el error.

1. Ejecutar la aplicación
2. Consultar la base de datos por nuevos registros
3. Abrir el browser
4. Leer noticias en yahoo.com
5. Cerrar el browser
6. Ir al mantenimiento de Clientes
7. Filtrar por Categoría “Nuevos Clientes”
8. Editar un cliente
9. Modificar datos, pero no grabar aún
10. Bajar el servidor de base de datos
11. Intentar grabar el cambio del cliente
12. Recibir un error de “timeout”
13. Cerrar la aplicación.

Resultado: el programa se cae.

En este ejemplo el tester transcribe todo lo que hizo antes de encontrar el error. Pero no se comprueba si todos los pasos, como leer las noticias de yahoo.com fueron necesarios.

Si el tester trabaja para detallar el número de pasos que son absolutamente necesarios, los desarrolladores son menos propensos a decir que el error no se puede reproducir y los Comités de Revisión de Errores son menos

propensos a decir que “nadie puede hacer todo esto”.

Pero ¿qué sucede si cada paso fuera necesario? Si los errores solo se manifiestan después de llevar a cabo todos los irrelevantes pasos. Debes escribir “paso necesario” ante tu aparente ilógico paso o debes agregar una nota al comienzo del reporte: “Todos los pasos descriptos aquí son necesarios para reproducir el error”. Escribiendo en forma clara los pasos muchas veces ayuda en el momento de verificar la solución del problema, especialmente si será otro tester quién realizará la verificación.

Explicar los efectos del error, no sólo los síntomas.

Algunos reportes de errores son engañosos. Una primera impresión indicaría que no hace mucho daño, pero si se examinan las implicaciones se pueden descubrir problemas muy serios. Si estuvieras en el Comité de Revisión de Errores ¿qué error priorizarías primero?

1- Un reporte que dice: “un molesto diálogo te impide cerrar la aplicación”

2- Un reporte que dice: “la aplicación se cuelga al salir”

El error es el mismo, la diferencia es la forma en que el tester lo reportó.

En este caso “un molesto diálogo” es la ventana que muestra Windows cuando no puede salir de un proceso. El tester encuentra el problema intentando apagar el equipo sin haber cerrado la aplicación. La aplicación no esperaba una entrada del usuario, por eso no había razón para que fallara en la salida. De hecho, este síntoma indica problemas más profundos, problemas que casi se pierden en el primer reporte (molesto diálogo que impide cerrar la aplicación).

Hay dos problemas con el “molesto” reporte del error. Primero, es impreciso. Si el tester hubiera incluido en su reporte “el molesto dialogo”, quiénes deben solucionar el error hubieran reconocido en el diálogo un serio problema más que una molestia menor. Segundo, el reporte no indica las consecuencias del error: la aplicación se cuelga.

Conclusiones

Todos pretendemos que nuestro trabajo marque la diferencia. Queremos que el sistema liberado sea mejor porque nosotros trabajamos en él. Que podamos hacer una diferencia o no estará muy influido por nuestra habilidad para comunicar los errores encontrados.

Cuando escribes un reporte de error, debes recordar a tu audiencia, elegir un buen título, registrar los pasos en forma clara y explicar las consecuencias del error. El reporte debe ser bueno por el esfuerzo extra que tú pones en él. Y cuanto mejor el reporte más chances de que la mayoría de los errores reportados sean solucionados.

Y en última instancia ese es el punto, tener más errores solucionados antes que puedan afectar a los usuarios.

Casos de prueba

Pruebas a partir de especificaciones

Para poder definir pruebas es necesario conocer las especificaciones (o requerimientos) del producto que vamos a probar. Estas especificaciones las podremos conocer a través del ESRE (Especificación de Requerimientos), las historias de usuario, o en algunos casos tendremos que preguntarle al cliente/equipo-de-proyecto/etc. ya que es posible que los requerimientos no estén formalmente definidos.

Para cada prueba tenemos que definir esto:

- Casos de prueba a partir de especificaciones
- qué hace el sistema que voy a probar
- objetivos de prueba
- resultados esperados
- pasos de ejecución
- condiciones iniciales

Para una especificación hay muchos casos de prueba con distintos datos de entrada

Ejemplo TV

Ejemplo:

- Especificación: "al apretar la tecla para subir el volumen se escucha más alto".
- Test: "apretar la tecla de subir el volumen y escuchar a ver si está más alto”.
- ¿Cuál es el valor esperado? que se escuche más alto.
- ¿Cuál es el objetivo de la prueba? verificar que aumenta el volumen.

Otro ejemplo:

- Al apretar el botón de prender – prende.
- ¿Pero si estaba prendida?
- ¡Hay que aclarar que hay que comenzar con la TV apagada!
- Esto muestra la importancia de especificar las condiciones iniciales.

Si aprieto el número 4, debería poner el canal 4. Si ahí aprieto el de subir entonces debería ir al 5. Con esto se ve la importancia de los **datos de prueba**.

Componentes de un **caso de prueba**. Partes importantes:

- Identificador del caso de prueba, único
- Título / objetivo
- Pasos
- Datos de entrada
- Valores esperados
- Condiciones iniciales
- Tester / equipo / responsable
- Fecha de creación
- Aplicación que se está probando

Diseño de Casos de Prueba

Nota: Al diseñar casos de prueba, hay 2 tipos:

- **Abstracto** - No define los juegos de datos específicos, sino que esto queda a criterio del tester que está ejecutando qué datos usar. Veremos que hay técnicas para elegir estos datos que hacen que el test tenga más o menos sentido. Ejemplo: “Caso de prueba para validar que el correcto funcionamiento del buscador”.
- **Concreto** - Es como un guión paso a paso dónde están indicados los datos de la prueba y se espera que el tester no se salga del mismo. En este caso, el diseñador de las pruebas es quien selecciona los datos y hará tantos casos de prueba como juegos de datos considere necesarios para evaluar la funcionalidad. Ejemplo: “Caso de prueba para buscar un producto que exista”.

Las partes de un caso de prueba

Un caso de prueba tiene varios campos. Como algunos no se usan siempre vamos a tratar de explicar los que a nosotros nos parecen más importantes y tal vez algunos opcionales que sean interesantes.

Campos obligatorios

ID (identificador)

Nos sirve para distinguir al caso de prueba de forma muy rápida, es algo así como el DNI del mismo, algo único que no se repite y que identifica al caso de prueba.
En Nahual, muchas veces simplemente ponemos 1, 2, 3...etc porque es una forma muy sencilla.
Cuando los proyectos son más grandes, generalmente están divididos por categorías, que a veces se conocen como módulos y cada uno tiene un prefijo. Por ejemplo, CO-001 es un id de un proyecto, y representa que es el caso 001 del módulo de Compras identificado como CO.
¿Para qué nos sirve?
Cuando queramos buscar ese caso de prueba que está linkeado a un bug, podemos buscarlo fácilmente por su ID.

¿Los casos de prueba están ordenados por id?

Esta es una pregunta muy común pero siempre surge en las clases cuando hablamos de ids. A veces en Nahual, donde probamos con aplicaciones chiquitas, los casos de prueba siguen un orden pero esto no pasa siempre y NO hay que acostumbrarse a que sea siempre de esta manera. Muchas veces el caso de prueba con id C-2 es justo el que se ejecuta luego del caso de prueba con id C-1 y así. Pero cuando las aplicaciones tienen sistemas más grandes casi nunca funciona así. Porque los proyectos van creciendo y vamos haciendo muchos casos de prueba en diferentes momentos. Entonces quizás el C-2 se ejecutaba luego del C-1 pero después tuvimos que

agregar casos de prueba en el medio porque se desarrolló algo nuevo durante el flujo. Y ahí se vuelve todo un lío porque deberíamos modificar todos los IDs de los casos de prueba que siguen y así. Esa forma de ordenar los casos de prueba no es escalable y cuando empiece a crecer la aplicación nos va a traer más dolores de cabeza que alivios.

Entonces, volviendo a la pregunta ¿Tienen que estar ordenados por id? No necesariamente, cada vez que vamos a ejecutar, tenemos que planear el orden pero no podemos pensar que siempre el 4 es el que viene después del 3. Los casos de prueba tienen que pensarse de forma atómica y de alguna manera independiente.

Título

Representa el objetivo de la prueba, tiene que ser fácil imaginarse de qué se trata cuando uno lo lee. Cuánto más conciso y claro, nos va a ayudar a la fácil lectura y compresión del mismo.

Un error muy común es utilizar frases como “verificar”, “probar”, “chequear”, “validar”, etc . Es cierto que en el fondo muchas veces queremos hacer esto, pero es redundante. Porque justamente volvemos a nombrar nuestro rol principal (verificar, probar) y no puntualizamos sobre qué se va a probar.

La forma correcta es siendo claros sobre lo que se va a llevar a cabo.

Aunque parece muy fácil, a veces encontrar el título ideal para el caso de prueba cuesta un poco. No se preocupen, eso se agiliza cuando vamos adquiriendo práctica y experiencia. Uno siempre aprende equivocándose. Borrando y volviendo a escribir, asi que no se estresen y en los primeros casos de prueba no puedan encontrar EL título, es algo que se va a ir puliendo a lo largo de todo el curso.

Si todavía tenemos dudas, vamos a tratar de explicarlo con algunos ejemplos:

Lo que NO debo hacer:

Verificar los números negativos.

Esto parece más una nota nuestra para acordarnos qué probar.

Si leemos esto luego de 3 meses, no entendemos QUÉ queremos verificar de los números negativos. ¿En qué parte de la aplicación lo pruebo? Si ingreso números negativos, ¿Qué estoy probando? ¿Qué no me los tome? ¿Qué me aparezca un mensaje de error? ¿Qué la aplicación convierta números negativos hasta cierto valor?

Luego de hacerme todas estas preguntas, ya pasaron como 30 minutos y tuve que leer todo el caso de prueba para entenderlo. Esa no es la idea, tenemos que tratar que nuestro trabajo sea lo más eficiente posible y una manera es tener un buen título en el caso de prueba.

Buenas prácticas, Buenos ejemplos:

Convertir números negativos: Para probar en Celfar los números que están por debajo del cero.
¿Por qué este es un buen ejemplo?

Si leo “Convertir números negativos” ya conozco la acción y la funcionalidad que estoy probando: la conversión. En este caso Celfar es una aplicación chiquita que tiene como objetivo satisfacer la necesidad de convertir valores que estén expresados en Celsius, mostrar los valores en Farenheit para ese dato.

Parece mentira, pero es mucho más claro que decir verificar porque la verificación se podría realizar de mil maneras distintas.

Me contesta el dónde, en la parte de la aplicación que se encuentre encargada de convertir.
Y el objetivo sería convertir números negativos, no hay valores límites.

Convertir números menores al cero absoluto: Acá vamos a probar los números por debajo del mínimo que define las especificaciones.

Esto es lo mismo que el ejemplo anterior. Me dice el qué, el dónde, y cuál sería el objetivo del caso de prueba: “la conversión de números menores al cero absoluto”.

Pasos a seguir

Es el corazón para la EJECUCIÓN de la prueba.
En esta instancia, indicamos el paso a paso de lo que hay que hacer en el momento de la ejecución. Para esto debemos ser muy claros, específicos y tratamos de que sea muy intuitivo y simple para la persona que los va a ejecutar (dado que no necesariamente va a ser uno mismo).
La mejor forma de hacerlo es enumerando o indicando mediante viñetas cada paso.
Es importante indicar en cada uno con qué dato estamos probando, se lo puede agregar en conjunto con cada paso o mencionarlo en una columna a parte (Columna de Datos). Pero es necesario que el dato esté. Leer luego buenas prácticas para esta instancia

Lo que NO debo hacer:

Título: C1-Convertir un número positivo
Pasos:
Ingresar un número
Convertir

¿Qué tienen de incorrecto estos pasos?
No son suficientemente específicos.
Ingresar un número puede ser cualquier número, la persona que se encargue de ejecutar las pruebas puede no darse cuenta de que número debe ingresar.
A su vez “Convertir” tampoco es claro, porque puede que la persona que ejecuta las pruebas no sepa cómo se lleva a cabo la conversión. Este es más bien un título (aunque sigue siendo incompleto) que un paso. Imaginemos que la forma de convertir no sea solamente apretando un botón llamado “Convertir”. Si hay otra forma de convertir, la persona que se encargue de ejecutar la prueba, no va a saber cómo hacerlo.

Veamos otra manera muy parecida de escribir lo mismo pero que soluciona estos errores:

Lo que DEBO hacer:

Título: C1-Convertir un número positivo
Pasos:

- Ingresar el número 3
- Apretar el botón convertir

En estos ejemplos de pasos es más claro lo que tiene que hacer la persona que ejecuta las pruebas. En caso de tener los pasos en una columna y los datos en otra este ejemplo resultaría:

Pasos	Datos
Ingresar un número entero Apretar el botón convertir	3

Resultado esperado

Cuando diseñamos un caso de prueba lo hacemos para probar que funcione algo. Pero cuando ejecutamos, ¿cómo nos damos cuenta de eso?

La forma en que visualiza ríamos o confirmaremos que el caso de prueba tuvo un resultado positivo es lo que completamos en RESULTADO ESPERADO.

Es muy difícil no caer en la tentación de copiar de la especificación cosas como: “que los datos ingresados se guarden correctamente” porque si bien es cierto que queremos que pase eso (porque así lo menciona la especificación), el resultado esperado tiene que ser comprobable.

Esto suele ser complicado de identificar dado que el hecho de que sea “comprobable” muchas veces no es trivial. Recordar que la diferencia radica en que la especificación nos indica cómo se debe comportar el sistema en forma general, Mientras que cada caso de prueba verifica sólo una funcionalidad específica del mismo. Por lo tanto, en el Resultado Esperado necesitamos detallar específicamente qué debemos ver en la aplicación al ejecutar ese caso y cómo lo debemos ver.

Ejemplo: Queremos probar una aplicación dónde queremos agregar un contacto a una agenda.
Veamos los siguientes errores comunes que encontramos en los cursos:

Ejemplos incorrectos de resultado esperado:

Que suceda lo que dice la especificación: Esto lo leemos mucho y es muy tentador escribirlo así, pero la idea es que cuando estamos ejecutando los casos no deberíamos tener que ir hasta la especificación. Nunca un resultado esperado tiene que estar escrito de esta manera.

Que el contacto se guarde correctamente: es muy tentador escribir que queremos que se guarde correctamente pero la persona que está ejecutando las pruebas, ¿cómo sabe que efectivamente se guardó correctamente? Tenemos que ser más claros con esta persona que está probando para que le resulte fácil identificar si efectivamente se guardó correctamente, y de esta manera saber si debe reportar un bug o no.

Para poder escribir un buen resultado esperado, tenemos que pensar cómo podría hacer la persona que va a ejecutar el caso de prueba para darse cuenta de que salió todo como se esperaba.

Ejemplo correcto de resultado esperado:

En la grilla de contactos figura el nombre Ariel con mail ariel@nahual.org: La forma de saber que un contacto se guardó correctamente es que aparezca en la grilla de los contactos.

Como vemos, es cierto que “sucedió lo que decía la especificación” y que “se guardó correctamente” pero lo importante es que la única forma de saber que el resultado es el correcto es viendo efectivamente dicho contacto (de nombre Ariel y mail ariel@nahual.org) en la grilla de contactos que es lo que muestra la pantalla de la aplicación.
Campos opcionales:
Descripción

El nombre lo dice todo, el foco está en poder explicar mejor de qué se trata la prueba, que sea más claro y fácil de entender para las personas que están por ejecutarla. Este campo suele ser opcional dado que muchas veces con el título alcanza pero en proyectos grandes esta casi siempre porque pasa lo contrario, solamente con un título no se logra explicar el objetivo del caso de prueba. En estos casos es OBLIGATORIO este campo ya que SOLAMENTE el título NO basta.

Datos (Opcional)

Si en los pasos no especificamos los datos con los cuales probar, se pueden indicar en una columna aparte.

Esto es muy útil en los proyectos grandes donde se hacen las mismas pruebas pero con varios datos diferentes.

Es una ventaja tener separados ambos campos ya que nos da flexibilidad, sólo debemos cambiar los datos y no tocamos los pasos. Este campo le da un valor extra al caso de prueba porque permite que sea más reutilizable.

Esto quiere decir que si algún dato se cambió, podamos modificarlo directamente desde esta columna y los pasos no se encuentren afectados por este cambio.

Precondiciones (Opcional)

Son las “condiciones” necesarias que debemos tener en cuenta para poder ejecutar la prueba, es decir, información que debemos tener de antemano.

Ejemplo 1: para poder editar un contacto, antes debemos contar con ese dato ya almacenado.
Ejemplo 2: en el caso que una app tenga varios perfiles de usuarios y se está probando una funcionalidad que sólo puede realizar el rol Administrador, por ejemplo, se deberá indicar el usuario con el cual loguearse para ejecutar dicha prueba.
Es decir que en esta columna se detalla específicamente el dato o escenario necesario para que se lleve a cabo la ejecución de la prueba. En el ejemplo 1 el contacto a editar y en el ejemplo 2 el usuario con el cual loguearse.

Podemos Concluir que los campos que deben estar en un caso de prueba son:

ID
Título
Descripción (Opcional cuando la aplicación es chiquita)
Pasos
Datos
Resultado Esperado

Tipos de datos / Testing Positivo y Negativo

Dato: Es un elemento aislado, recabado para un cierto fin, pero que no ha pasado por un proceso que lo interrelacione con otros de manera funcional para el fin previsto.

- Ejemplo de datos: 20, Juan, cédula.

Información: Se trata del conjunto de datos, añadidos, procesados y relacionados, de manera que pueden dar pauta a la correcta toma de decisiones según el fin previsto.

- Ejemplo de información: La cédula de juan indica que tiene 20 años (Los datos se relacionan para indicar algo, aislados no significan nada; en esta oración indican algo).

Fuente: http://www.informaticamoderna.com/Info_dat.htm

Boolean (booleano): Este tipo de dato se emplea para valores lógicos, los podemos definir como datos comparativos dicha comparación devuelve resultados lógicos (Verdadero o Falso).

Char (caracter): El tipo de dato carácter es un dígito individual el cual se puede representar como numéricos (0 al 9), letras (a-z) y símbolos (!"$&/\).

String (cadena): Es un conjunto de caracteres, o sea un texto. Ejemplos: “Hola mundo”, “Vale 100 $”.

Número: Este tipo de dato puede ser real o entero, dependiendo del tipo de dato que se vaya a utilizar.

- Enteros: son los valores que no tienen punto decimal, pueden ser positivos o negativos y el cero.
- Reales: estos caracteres almacenan números muy grandes que poseen parte entera y parte decimal.

Fecha: Se manejan distintos tipos de datos según el lenguaje de programación o base de datos para almacenar fechas, lo importante es que tienen distintos formatos aceptados. Ejemplo: “02/10/2015” formato clásico dd/mm/aaaa (día/mes/año).

Multimedia: El término **multimedia** se utiliza para referirse a cualquier objeto o sistema que utiliza múltiples medios de expresión físicos o digitales para presentar o comunicar información. De allí la expresión multimedios. Los medios pueden ser variados, desde texto e imágenes, hasta animación, sonido, video, etc. También se puede calificar como *multimedia* a los medios electrónicos u otros medios que permiten almacenar y presentar contenido multimedia. Multimedia es similar al empleo tradicional de medios mixtos en las artes plásticas, pero con un alcance más amplio.

Fuentes:

- https://es.wikipedia.org/wiki/Cadena_de_caracteres
- https://es.wikipedia.org/wiki/Tipo_de_dato
- https://es.wikipedia.org/wiki/Multimedia

Test Positivo (o limpio): Intenta mostrar que el producto satisface sus requerimientos.

- ¿Cómo hago un test positivo? R: Armo mi test con el requerimiento que aparece en la especificación.
- Ejemplo: Para el carrito pide que se puede ingresar el nombre y el apellido del cliente, hago mi caso para ingresar el nombre Juan y apellido Pérez (un nombre y apellido cualquiera).

Test Negativo (o sucio): El objetivo es romper el sistema.

- ¿Cómo hago un test negativo? R: Trato de hacer algo contrario a lo que el sistema espere que se haga.

- Ejemplo: Para el mismo requerimiento del carrito de nombre y apellido, tratar de poner un número en lugar de un nombre y un apellido. El resultado debería ser que se muestre un aviso de error (nunca debería caerse el sistema e impedir que siga registrando el usuario).

Fuente: http://www.practicasdesoftware.com.ar/2011/02/testing-y-algo-mas/#.VgQrMd94vQo

Cobertura de pruebas

No es posible probar todo

Si nos preguntamos ¿cuántas pruebas podríamos ejecutar para una funcionalidad? ... depende de muchas variables:

- Cuántos datos tengo que poner en cada pantalla
- Cuántos valores distintos puedo poner en cada campo de la pantalla
- ¡¡¡cómo combinar esos datos!!!
- Además, puedo ejecutar en distintos navegadores, en distintos celulares, sistemas operativos, etc., etc.

Entonces la respuesta es: ¡infinitos casos!
Claramente, no tenemos tiempo para probar todo.

La forma en la que los testers estaremos aportando calidad al software es principalmente buscando fallos. Por supuesto. Digamos que si no encuentro fallo todo el costo del testing me lo pude haber ahorrado. ¿No? ¡Nooo! Porque si no encontramos fallos entonces tenemos más confianza en que los usuarios encontrarán menos problemas.

¿Se trata de encontrar la mayor cantidad de fallos posible? ¿Un tester que encuentra cien fallos en un día hizo mejor su trabajo que uno que apenas encontró 15? De ser así, la estrategia más efectiva sería centrarse en un módulo que esté más verde, que tenga errores más fáciles de encontrar, me centro en diseñar y ejecutar más pruebas para ese módulo, pues ahí encontraré más fallos.
¡NO! La idea es encontrar la mayor cantidad de fallos que más calidad le aporten al producto. O sea, los que al cliente más le van a molestar.

¡Ojo!, el objetivo no tiene por qué ser el mismo a lo largo del tiempo, pero sí es importante que se tenga claro en cada momento cuál es. Puede ser válido en cierto momento tener como objetivo del testing encontrar la mayor cantidad de errores posibles, entonces seguramente el testing se enfoque en las áreas más complejas del software o más verdes. Si el objetivo es dar seguridad a los usuarios, entonces seguramente se enfoque el testing en los escenarios más usados por los clientes.

Existen distintas técnicas de diseño de casos de prueba, que permiten seleccionar la menor cantidad de casos con mayor probabilidad de encontrar fallas en el sistema. Por este motivo, estos casos se consideran los más interesantes para ejecutar, ya que **el testing exhaustivo (ver definición o ejemplo) no solo es imposible de ejecutar en un tiempo acotado, sino que también es muy caro e ineficiente.** Es así que se vuelve necesario seleccionar de una forma inteligente los valores de entrada que tengan más posibilidades de descubrir un error.

¿Qué es la Cobertura de pruebas?

Básicamente, es una medida de calidad de las pruebas. Se definen cierto tipo de entidades sobre el sistema, y luego se intenta cubrirlas con las pruebas. Es una forma de indicar cuándo probamos suficiente, o para tomar ideas de qué otra cosa probar (pensando en aumentar la cobertura elegida).

Para verlo aún más simple, podríamos decir que la cobertura es como cuando barremos la casa. Siempre se me olvida el cuarto, eso es que en mi barrido no estoy cubriendo el cuarto. Mide la calidad de mi barrido, y a su vez me da una medida para saber cuándo tengo que terminar de barrer: cuando cubra cada habitación, por ejemplo.

Ahora, lograr el 100% de cobertura con ese criterio, ¿indica que la casa está limpia?
NO, porque la cocina y el comedor ¡ni los miré! Entonces, ¡ojo!, manejar el concepto con cuidado. Tener cierto nivel de cobertura es un indicador de la calidad de las pruebas, pero nunca es un indicador de la calidad del sistema, por ejemplo, ni me garantizará que está todo probado.

¿Entonces para qué me sirve?

- Medida de calidad de cómo barro
- Me indica cuándo parar de barrer
- Me sugiere qué más barrer

Unos criterios pueden ser más fuertes que otros, entonces el conocerlos me puede dar un indicador de qué tan profundas son las pruebas, cuándo aplicar uno y cuándo otro.

- Criterio 1: barrer cada habitación.
- Criterio 2: barrer cada pieza (habitaciones, comedor, cocina, baño, etc.).
- Criterio 3: barrer cada pieza, incluso en las esquinas, porque ahí hay más posibilidades de que se acumule suciedad.

Testing de Regresión

Las pruebas de regresión son algunas de las pruebas que tengo diseñadas que se seleccionan para ejecutar periódicamente, por ejemplo, ante cada nueva versión del producto.

Tienen el objetivo de verificar que el producto no ha sufrido regresiones. Entonces, cada vez que hay una nueva versión del software, nosotros los testers vamos a decidir ejecutar las mismas pruebas que tenemos diseñadas, y que nos aseguran que las funcionalidades que el software tiene siguen funcionando.
Por ejemplo, en el carrito, en cada nueva versión deberíamos probar:

- que puedo registrar un usuario
- que puedo agregar elementos al carrito y se calcula bien el precio
- que puedo pagarlo

Son funcionalidades muy importantes, y si estas no funcionan, entonces el sistema va a darle problemas a los usuarios, y de ahí la importancia de ejecutarlas cada vez.

¿Por qué se llaman pruebas de regresión?
En un principio pensaba que se refería a regresar a ejecutar las mismas pruebas, ya que se trata un poco de eso. Luego vi que el concepto en realidad está asociado a verificar que lo que estoy probando no tenga regresiones. El tema es que “no tener regresiones” imaginé que se refería a que no haya una regresión en su calidad, o en su funcionalidad, pero escuché el rumor de que el concepto viene de la siguiente situación: si los usuarios tienen la versión N instalada, y le instalamos la N+1, y esta tiene fallos, nos veremos atormentados al tener que volver a la versión previa, hacer una regresión a la versión N. ¡Queremos evitar esas regresiones! Y por eso se realizan estas pruebas.

Tampoco es válido pensar que las pruebas de regresión se limitan a verificar que se hayan arreglado los bugs que se habían reportado, pues es igual de importante ver que lo que antes andaba bien ahora siga funcionando.

Generalmente cuando se diseñan las pruebas para determinadas funcionalidades, ya se definen cuáles son las pruebas que serán consideradas dentro del conjunto de pruebas de regresión. O sea, las que serán ejecutadas ante cada nueva liberación del producto, o en cada ciclo de desarrollo. Ejecutar las pruebas de regresión consistirá en ejecutar nuevamente las pruebas previamente diseñadas.

Hay quienes dicen que al contar con una listita con los pasos a seguir y las cosas a observar no se está haciendo testing, sino un simple chequeo. James Bach y Michael Bolton, dos de los expertos en testing, comentan en un par de artículos las diferencias entre el testing y el chequeo. El testing es algo donde uno pone creatividad, atención, busca caminos nuevos, piensa “¿de qué otra forma se puede romper?”. Al chequear simplemente, nos dejamos llevar por lo que alguien ya pensó antes, por esa ya mencionada lista de pasos.

Un problema que surge con esto es que las pruebas de regresión viéndolas así son bastante aburridas. Aburrimiento genera distracción. La distracción provoca errores. El testing de regresión está atado al error humano. ¡Es aburrido volver a revisar otra vez lo mismo! eso hace que uno preste menos atención, e incluso, puede llegar a darse la situación de que uno desea que algo funcione e inconscientemente confunde lo que ve para dar un resultado positivo.

¡Ojo! ¡No estamos diciendo que el testing sea aburrido! Al menos a nosotros nos encanta. Estamos diciendo que las cosas rutinarias son aburridas y, por ende, propensas al error. Esto hace que tengamos que tener especial cuidado al ejecutarlas.

Ejemplo de reporte

15 de octubre de 2015

Reporte de estado de producto

Super X - Carrito v3

Estimado cliente,

De acuerdo a lo conversado en nuestra última reunión, detallamos el estado actual del proceso de testing de la versión 3 del Carrito.

Al igual que para las versiones anteriores, las pruebas realizadas por el equipo de testing de Nahual incluyen:

- prueba funcionales sobre los nuevos requerimientos
- pruebas de regresión de todos los casos de prueba existentes para la versión 2

Set de casos y resultados

	Casos totales	**Aprobados**	**Fallidos**	**No Ejecutados**
Grupo 1	8	6	2	0
Grupo 2	10	5	3	2
Grupo 3	8	0	0	8
Total	**26**	**11**	**5**	**10**

Próximos pasos

Para terminar con los trabajos pendientes antes de la fecha prometida el equipo seguirá la siguiente estrategia:

1. el equipo de desarrollo se comprometió a extender su horario en 10hs por semana para solucionar los bugs reportados dentro del próximo mes
2. el equipo de testing se comprometió a terminar los casos de prueba pendientes dentro de los próximos 5 días de forma de dar tiempo a desarrollo para corregir los nuevos bugs que puedan aparecer.

Quedamos a las órdenes para aclarar cualquier consulta que pueda surgir.
Atentamente,
Equipo Nahual

Clases de equivalencia

Motivación

Dado que probar todo es imposible, y en realidad del universo de entradas posibles vamos a poder probar solo un subconjunto, queremos elegir aquel subconjunto que tenga la mayor probabilidad posible de encontrar errores.

El desafío: ¡probar lo más posible con el menor esfuerzo posible! :)

Técnica de diseño

“Partición en Clases de Equivalencia” es una técnica de diseño de casos de prueba de **Caja Negra (Ver Anexo)**.

Pasos de la técnica:

1. Identificar las clases de equivalencia
2. Definir los casos de prueba

Paso 1: Identificar las clases de equivalencia

Para identificar las clases de equivalencia, se consideran los valores de **entrada** y los valores de **salida** del programa bajo prueba.

Se definen 2 tipos de clases de equivalencia: clases **válidas** y clases **inválidas**. Las cuales corresponden a entradas válidas e inválidas, respectivamente.

Para empezar:

- Identificamos las variables existentes y sus posibles valores.
- Identificamos las clases de equivalencia.

¿Cómo identificar las clases de equivalencia?

Pasos:

1. Si la condición de entrada corresponde a un rango de valores *(por ejemplo: “un número entre 1 y 999”)* → se define 1 clase de equivalencia válida (CEV) y 2 inválidas (CEI). En el ejemplo:

a. *CEV: 1 <= valor <= 999*

b. *CEI: valor < 1 y valor > 999*

2. Si la condición de entrada corresponde a un valor específico *(por ejemplo: “el primer carácter tiene que ser una letra”)* → 1 CEV *(una letra)* y 1 CEI *(no es una letra).*

3. Si la condición de entrada corresponde a un conjunto de valores posibles *(por ejemplo: “el tipo de vehículo puede ser {moto, auto, camión}”)* → se define una CEV **para cada uno** de los valores válidos, y una CEI *(ejemplo: bicicleta)* para un valor no esperado.

4. Si la condición de entrada especifica el número de valores *(por ejemplo: “una casa puede tener de uno a seis propietarios”)* → definir 1 CEV y 2 CEI *(0 propietarios y más de 6 propietarios)*.

Si tenemos la duda de que el software bajo prueba no trata a todos los elementos de una clase de equivalencia

en forma idéntica, entonces hay que **subdividir** la clase de equivalencia en subclases más chicas.

Paso 2: Definir los casos de prueba

Procedimiento:

1. Asignar un identificador único a cada clase de equivalencia.

2. Definir casos de prueba hasta contemplar todas las clases de equivalencia válidas. Intentando incluir la mayor cantidad de clases válidas en cada caso de prueba *(de modo de minimizar la cantidad de casos de prueba)*.

3. Para las clases de equivalencia inválidas, definir un caso de prueba por cada clase de equivalencia inválida.

a. Notar la diferencia con el paso anterior, ¿por qué la diferencia de criterio? porque queremos evitar enmascarar errores! O sea, si mezclamos varias clases de equivalencia inválidas y obtenemos un error (Ok, es el resultado esperado) pero no podemos confirmar cuál fue el valor que originó el error. No estamos seguros de que cada entrada que esperamos ocasione un error, realmente lo provoque.

Por ejemplo (valga la redundancia), se puede usar el ejemplo que presenta acá:

http://www.bstriker.com/3encuentro_Tecnicas.pdf

Referencias

- "The Art of Software Testing" by Glenford J. Myers.

Análisis de condiciones de borde

Variación de la técnica de partición de equivalencias, que se focaliza en los bordes de cada clase de equivalencia: por arriba y por debajo de cada clase.

¿Por qué? R: Porque los programadores usan condiciones matemáticas (>, >=, <, <=), y es fácil que se equivocan y pongan mayor en lugar de mayor o igual, y asi como con las demás condiciones.

Ejemplo: Mirando el ejemplo anterior vemos que se acepta un número entre 1 y 999. Vemos que hay 3 clases de equivalencia. Una válida *CEV: 1 <= valor <= 999* y dos inválidas, *CEI1: valor < 1,* y *CEI2: valor > 999.* Tengo que elegir al menos un elemento de cada clase para probar, pero para hacerlo de forma inteligente aplico condiciones de borde. Entonces tomo los valores 0 (Mayor elemento de CEI1), 1 (Menor elemento de CEV), 999 (Mayor elemento de CEV), y 1000 (Menor elemento de CEI2).

Referencias: http://es.slideshare.net/agrosso/taller-casos-de-prueba

Árboles de decisión

Introducción

¿Cómo surge la necesidad?

Existen condiciones en las diferentes aplicaciones (programas) que se deben cumplir y necesitamos organizarnos para no dejar cosas sin probar.

¿Se acuerdan de los casos de prueba que estuvimos trabajando?

Registro de clientes con los siguientes datos: Nombre y apellido, Fecha de nacimiento, Dirección, Email y contraseña, Teléfono. Hasta ahora solo probamos casos de prueba positivos y negativos, casos de borde y clases de equivalencia.

¿Cómo hacemos para probar todas las combinaciones?

Necesitamos una herramienta que nos ayude a asegurarnos de recorrer todos los caminos posibles.

En el diseño de aplicaciones informáticas, un árbol de decisión indica las acciones a realizar en función del valor de una o varias variables. Es una representación en forma de árbol cuyas ramas se bifurcan en función de los valores tomados por las variables y que terminan en una acción concreta. Se suele utilizar cuando el número de condiciones no es muy grande.

Veamos si queda clara la idea con un par de ejemplos.

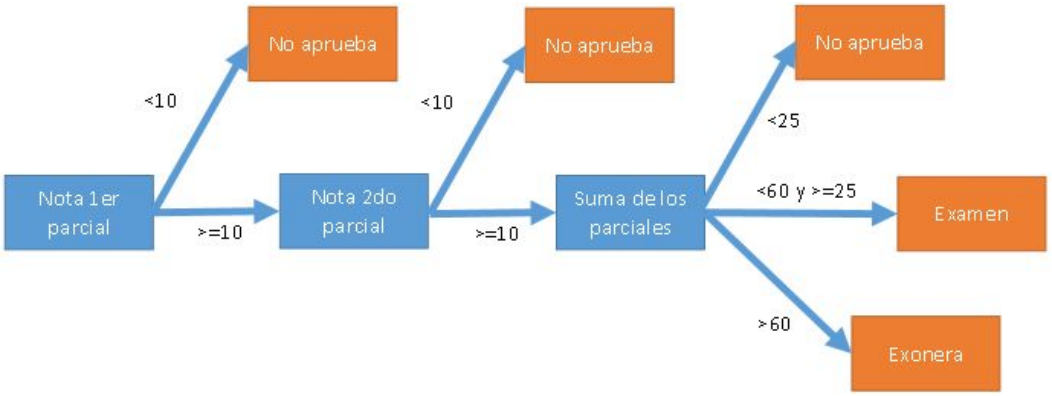
Ejemplo 1 - Parciales

Tenemos una aplicación con una pantalla que recibe las notas de dos parciales y muestra el resultado final del alumno al presionar el botón “Calcular”.

Debemos considerar las siguientes reglas:

- Primer parcial: 0-30
- Segundo parcial: 0-70
- En cualquiera de los dos casos, si sacan menos de 10 pierden el curso
- Con menos de 25 puntos en total, pierden el curso
- Entre 25 y 59 van a examen.
- Más de 60 aprueban el curso.

Una forma de representar esto con un árbol es como se muestra en la siguiente figura.



Esta representación nos permite visualizar fácilmente las combinaciones de datos interesantes, y seleccionar pruebas de forma tal de considerar todas las reglas, y no repetir situaciones o no dejar ninguna sin probar.

Datos de prueba

Nota 1er Parcial	**Nota 2º Parcial**	**Suma Notas**	**Valor Esperado**
9			No aprueba
10	9	19	No aprueba
10	14	24	No aprueba
10	15	25	A Examen
30	30	60	Exonera

Si usamos esos datos logramos cubrir el árbol de la figura. ¿Te animas a comprobarlo? Revisa y asegúrate que con cada juego de datos de la tabla quedan cubiertas todas las hojas del árbol.

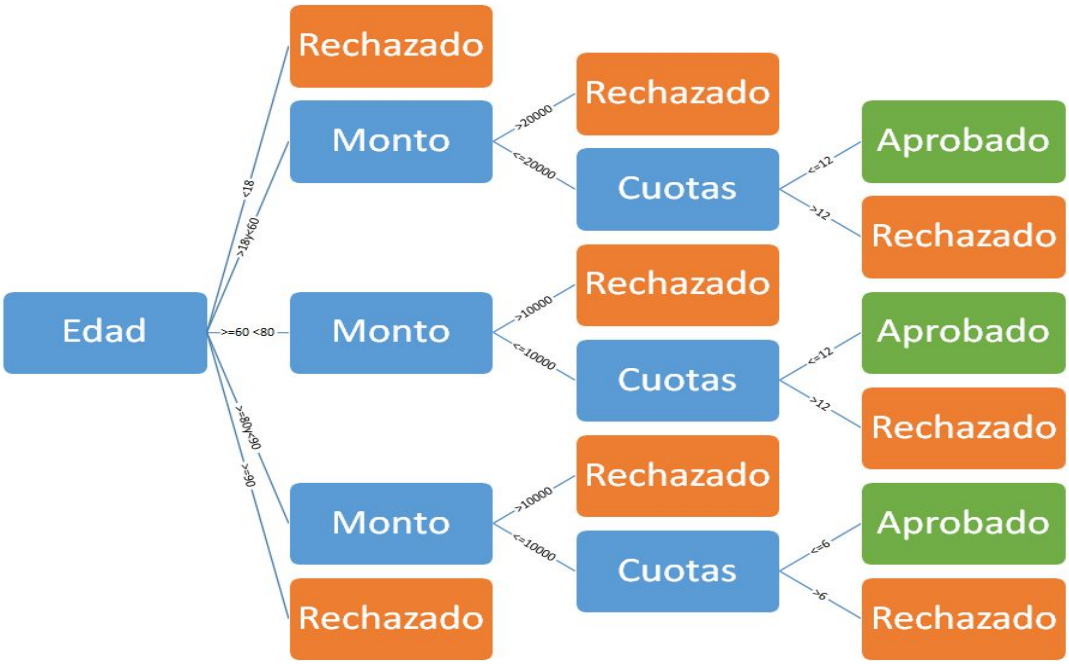
Ejemplo 2 - Préstamo

Un banco nos pide probar una aplicación que será la encargada de aprobar o rechazar préstamos a sus clientes.

La aplicación no está desarrollada, entonces los testers tienen que ir diseñando las pruebas para que cuando esté lista se pueda probar. Tienen que armar las pruebas de acuerdo a las siguientes condiciones para poder aprobar un préstamo:

- Hay que ser mayor de edad para que te den el préstamo.
- No se dan préstamos de más de 20.000 (no hay una prueba donde esté en el rango de edad, pero pida más dinero).
- Si sos mayor de 60, el tope es 10.000 pesos.
- No se dan más de 12 cuotas.
- No se dan préstamos de más de 6 cuotas a personas mayores de 80.
- No se dan préstamos a personas mayores de 90.

Estas reglas de negocio las podríamos representar en forma de árbol como puedes observar en la siguiente figura. Por favor revisa que están todas contempladas.



Luego, basándonos en todas las ramas, y buscando cómo visitar todas las hojas, diseñamos los datos de prueba tal como se muestran en la siguiente tabla:

Datos de prueba

edad	**monto**	**cuotas**	**valor esperado**
17			rechaza préstamo
18	15.555.555		rechaza préstamo
18	15.000	15	rechaza préstamo
20	15.000	5	aprueba préstamo
70	20.000		rechaza préstamo
70	10.000	15	rechaza préstamo
70	10.000	12	aprueba préstamo
80	15.000		rechaza préstamo
80	10.000	10	rechaza préstamo

80	10.000	6	aprueba préstamo
90			rechaza préstamo

Testing exploratorio

Si vamos a planificar un viaje a una ciudad que no conocemos, tenemos dos formas:

- compramos un mapa y una guía y hacemos un plan de lo que queremos visitar
- o bien compramos el mapa y la guía y los vamos leyendo en el lugar.

Cuando planificamos tenemos que dedicar tiempo a planificar, pero es más fácil luego compartir el plan con un amigo para que haga otra vez lo mismo, y sabemos decirle cuánto tiempo le llevaría.

En la segunda opción, uno va viendo y va resolviendo en el camino. Lo bueno es que no tenemos que dedicar tiempo a planificar, sino que planificamos y exploramos al mismo tiempo.

Básicamente el **testing exploratorio** es EJECUCIÓN y DISEÑO de las pruebas al mismo tiempo. ¡¡No se escriben casos de prueba!!

Esto es ideal para cuando vamos a probar un sistema que no conocemos, o que no tenemos mucho tiempo, o que no hay mucha documentación como para ir pensando las pruebas de antemano.

Las pruebas las hacemos en "**sesiones**", que son bloques de tiempo de unos 40 minutos, una hora, hora y media, a veces más, pero es importante que sea ININTERRUMPIDO. Por esto es que no deberían ser nunca de más de 2 horas.

Vamos a ver algunas de las cosas que tenemos que registrar para una sesión:

- **Misión**: en lugar de tener un caso de prueba que me dice paso por paso lo que tengo que hacer, acá vamos a tener misiones, como en los juegos de computadora que tenemos que conseguir ciertos elementos, tantas monedas de oro, etc., o como un explorador que tiene que recorrer cierto lugar y encontrar el Cáliz de Oro cual Indiana Jones. Si lo asociamos con el viaje a la ciudad podríamos poner como misiones:
- recorrer todas las plazas de la ciudad
- ir de la playa más al norte a la playa más al sur

Si lo pensamos en el sistema del Carrito Nahual:

- ver si un usuario nuevo puede comprar en el carrito
- intentar comprar con variedad de productos
- revisar las restricciones de edad para ciertos productos

- **Áreas:** Contiene la información sobre el cubrimiento de las áreas funcionales, las plataformas, datos, operaciones, y también, técnicas de testing a usar para el modelado de la aplicación o sistema bajo prueba.
- **Tester**: debemos registrar qué tester ejecuta cada sesión.
- **Tiempo de inicio y fin**
- **Duración**
  - corta, media, larga. Es importante que sea ininterrumpida, así que nunca deberían ser más de 2 hs, por más que sea la larga, ya que nos van a dar ganas de ir al baño, o ¡revisar Facebook!
  - entonces, esta duración se establece antes de comenzar, y deberíamos respetarla.
- **Archivos:** si se requiere algún archivo, se registra cuáles se usaron.
- **Métricas** de tiempo, o sea, datos numéricos, que nos dan información de cómo fue nuestra prueba. En

particular se suele tomar nota de lo siguiente:

- % de tiempo ejecutando y diseñando
- % de tiempo reportando bugs
- % de tiempo preparando la prueba

- **Métricas** de misión versus oportunidad, o sea, registrar cuánto tiempo estuvimos concentrados en la misión y cuánto tiempo nos fuimos por las ramas.

- **Errores:** se toman nota de los errores, sugerencias de mejora, cosas que no se entienden y luego habrá que discutirlas con alguien para ver si están bien o no, etc.

- **Notas de prueba:** la idea es que uno debería ir anotando todas las cosas que hizo, así luego cualquiera puede analizar qué cosas se probaron y qué cosas no.

- **Inconvenientes:** Inconvenientes o impedimento que haya surgido quitando tiempo a la sesión y la concreción de su objetivo.

Tal como cuando uno explora una ciudad, uno tal vez tiene una misión en la cabeza, va haciendo un recorrido que pensó hacer, pero en ciertos lugares le llama la atención algo y se desvía. En el testing exploratorio nos podemos desviar de nuestra misión si vemos algo que tal vez pueda descubrir un error, o algo nuevo, o algo que llamó la atención, pero es muy importante siempre volver a la misión principal, estar solo de a ratos yéndonos por las ramas, y siempre volver a nuestro objetivo.

Ejemplo de Sesión de Testing Exploratorio

Aplicación a probar: Como ir - IMM http://www.montevideo.gub.uy/aplicacion/como-ir

La IMM tiene transporte de un punto a otro de Montevideo

- independientemente de la zona,
- en todos los horarios,
- todos los días del año

por lo que la aplicación debería encontrarlos.

- **Misión**: Identificar posibles problemas de movilidad, porque los usuarios de la aplicación no encuentran ómnibus / camino (ir caminando) de un lugar a otro.

- **Áreas:** Funcionalidad de ver “cómo ir” de un punto a otro (caminando y en ómnibus). Plataforma Windows, navegador: Google Chrome.

- **Tester**: Fulano.

- **Tiempo de inicio y fin:** 10.30am hasta 11am.

- **Duración: corta, 30 min.**

- **Métricas** de tiempo:

- % de tiempo ejecutando y diseñando: 70%
- % de tiempo reportando bugs: 25%
- % de tiempo preparando la prueba: 5%

- **Métricas** de misión versus oportunidad:
  - 80% en misión
  - 20% en oportunidad
- **Errores:**
  - error número 1: aparece mensaje de error cuando...
  - error número 2: ...
- **Notas de prueba:**
  - Ingresamos a la aplicación, buscamos ir de un punto a otro dando esquinas
  - Probamos con calle y número
  - Probamos con lugares de interés
  - Probamos imprimir el mapa y las indicaciones
  - ...
- **Inconvenientes:**
  - No se entendió para qué servía el link a “ver demostración”.
  - Se tuvo problemas con ...

Propiedades del Testing Exploratorio

Algunos datos importantes sobre el testing exploratorio:

- Las pruebas no son definidas con anticipación, se espera que el tester aprenda con rapidez, acerca de un producto o nueva funcionalidad mientras que se provee retroalimentación al resto del equipo.
- Los resultados obtenidos durante pruebas anteriores guiarán las acciones, los pasos y los siguientes escenarios de prueba a ejecutar, construyendo así un conjunto de pruebas más eficaz.
- Su foco está en encontrar problemas y defectos por medio de la exploración del sistema bajo prueba.
- Es un acercamiento a las actividades de testing, que consiste en una serie de actividades que se realizan en simultáneo, tales como el aprendizaje del sistema; diseño y ejecución de las pruebas.
- La efectividad del testing se apoya en el conocimiento, habilidades y experiencia del tester.

Algunos puntos a considerar al armar una sesión serían los siguientes: **dónde** debemos enfocar nuestros esfuerzos durante nuestra exploración; **qué** recursos tenemos a nuestra disposición; **cuál** es la información que descubrimos durante nuestro trabajo.

- **Dónde**: El objetivo de nuestro trabajo de exploración. Podría ser posible que estemos trabajando probando un requerimiento, un módulo, una nueva funcionalidad, etc.
- **Qué**: Herramientas a utilizar. Pueden estar en cualquier forma, desde una nueva técnica, la ayuda de un compañero u otro actor que esté involucrado en el proyecto.
- **Cuál**: Contando con los puntos anteriores, debemos enfocarnos en encontrar información relevante que nos ayude a proporcionar valor al resto del equipo.

Análisis de resultados de la sesión

¿Para qué nos sirven las métricas?

- Si en las métricas vemos que estuvimos mucho tiempo reportando errores, significa que tenemos que dedicarle más tiempo a esta misión, quizá ejecutarla de nuevo.
- Si observamos que estuvimos mucho tiempo en “oportunidad” más que en “misión”, eso significa que para la próxima deberíamos dividir en dos misiones distintas, ya que hay muchas cosas para irnos por las ramas.

Este tipo de testing es ideal para ir conociendo una aplicación. Tal vez comenzamos con una misión genérica, luego nos damos cuenta que la tenemos que dividir en distintas partes, ya que fuimos conociendo el sistema, y así vamos progresando, refinando nuestro conjunto de pruebas exploratorias.

Luego con las notas de prueba, alguien podría crear casos de prueba considerando cuáles son las cosas más importantes que tenemos que ejecutar, para incluirlas en el **testing de regresión**.

Ejemplo de Sesión de Testing Exploratorio:

OBJETIVO	Revisar ventas de carrito con máximos de stock
ÁREAS	Sistema Ubuntu Versión X de carrito
INICIO	DD/MM/AAAA - HH:MM am/pm
TESTER/S	Equipo de testers
ESTRUCTURA DE DIVISIÓN DE TAREAS x DURACIÓN	DURACIÓN: Corta (45´) DISEÑO Y EJECUCIÓN DE PRUEBAS: 70% INVESTIGACIÓN Y REPORTES DE DEFECTOS: 15% ARMADO DE LA SESIÓN: 15% x OBJETIVO vs. OPORTUNIDAD: 85 / 15
ARCHIVOS DE DATOS	Documento_de_Especificaciones.pdf x Documento_de_Usuario.Administrador.pdf
NOTAS DE PRUEBAS	[Prueba #1] [Prueba #2] [Prueba #3]
DEFECTOS	ID #12345: ¡Hola! Soy un bug, y mi nombre es...
INCONVENIENTES	[#1] Desde las 9am y hasta las 9.40am el ambiente de pruebas no estuvo disponible, debido a inconvenientes con el servidor web. La sesión se pausó y se retomaron las actividades a partir de las 9.45am. [#2] Durante la ejecución de la Prueba #2 durante la sesión, el servidor arrojó errores de tipo HTTP-500. Dichos errores fueron reportados a infraestructura y desarrollo, para lo que se creó el ticket con ID #7563.

Métricas de Testing Exploratorio:

Sesión	Fecha	Hora	Dur.	Ob.	Op.	Test	Def.	Setup	#Bugs	#Iss	#T
et-jsb-010416-a	4/16/14	9:30 am	45’	1	0	0.8	0	0.2	0	3	1

Material Extra

Técnicas de Testing

- **Caja negra (o funcional):** Abarca a aquellos criterios que deciden si una suite es adecuada analizando la especificación del software a testear (pero no su código)
- **Caja blanca (o estructural):** Abarca a aquellos criterios que deciden si una suite es adecuada analizando la estructura del código a testear.
- **Referencias:** http://dc.exa.unrc.edu.ar/rio2013/sites/default/files/notas-03.pdf

Hardware y software

- http://www.gcfaprendelibre.org/tecnologia/curso/informatica_basica/empezando_a_usar_un_computador/1.do
- http://www.gcfaprendelibre.org/tecnologia/curso/informatica_basica/empezando_a_usar_un_computador/2.do

El tester y la imaginación

- http://testingbaires.com/el-tester-y-la-imaginacion/

El oficio del testing

- http://blog.abstracta.com.uy/2014/10/fabrica-olmos-mi-madre-trabajo-de-tester.html

Buenos reportes

- https://josepablosarco.wordpress.com/2010/12/09/%C2%BFcomo-escribir-un-buen-reporte-de-fallasbugs/
- http://algomasquetraducir.com/testing-de-videojuegos-y-software-los-informes-de-bugs/ (este en particular habla de trabajar haciendo testing de videojuegos y tiene las mismas consideraciones).
- http://m.genbetadev.com/metodologias-de-programacion/la-importancia-de-escribir-correctos-informes-de-bugs

Proceso de creación de Software

- http://testingbaires.com/pasos-principales-de-las-pruebas-de-calidad/

General

- Software Holmes: http://www.cromo.com.uy/software-holmes-n596226
- ¿Los testers deben saber hacer de todo en el mundo ágil? ¿Qué cualidades debe tener un tester en un equipo ágil?: http://www.javiergarzas.com/2015/05/debate-tester-agil.html

Bugs viejos, prioridad y severidad

- http://www.neoteo.com/microsoft-corrige-un-bug-de-15-anos-en-windows/

Bugs famosos

- Los peores "'bugs" (errores informáticos) de la Historia: http://axxon.com.ar/not/156/c-1560164.htm
- 10 históricos errores tecnológicos: http://hipertextual.com/2007/11/10-historicos-errores-tecnologicos
- Los 10 más grandes errores de la Informática: http://listas.20minutos.es/lista/los-10-mas-grandes-errores-de-la-informatica-337766/
- Los 20 desastres más famosos relacionados con el Software: http://www.taringa.net/posts/info/1890933/Los-20-desastres-mas-famosos-relacionados-con-el-Software.html
- La "Mars Climate" se estrelló en Marte porque la NASA no tradujo kilómetros a millas:

http://elpais.com/diario/1999/10/02/sociedad/938815207_850215.html
- Malos tipos de cambio: http://www.montevideo.com.uy/auc.aspx?270908%2C3%2CASOCIADAS_ABAJO
- El Gangnam Style rompió YouTube: http://www.elobservador.com.uy/el-gangnam-style-rompio-youtube-n293451

Recompensas por encontrar bugs

- United Airlines will let you fly free if you find bugs in its software: http://uk.businessinsider.com/united-airlines-bug-bounty-free-air-miles-flight-security-vulnerability-2015-5#ixzz3l3VO84HK
- Google paga hasta 38000 dólares a quien vea bugs en Android: http://computerhoy.com/noticias/moviles/google-paga-38000-dolares-quien-vea-bugs-android-29919
- Dropbox ofrecerá recompensas por cazar fallos de seguridad: http://www.siliconweek.es/cloud/dropbox-ofrecera-recompensas-por-cazar-fallos-de-seguridad-79202

Búsqueda laboral

- Redes sociales: el nuevo jugador en la búsqueda laboral: http://www.elobservador.com.uy/redes-sociales-el-nuevo-jugador-la-busqueda-laboral-n222491
- Fue despedida antes de su primer día de trabajo por Twitter: http://www.subrayado.com.uy/Site/noticia/41820/fue-despedida-antes-de-su-primer-dia-de-trabajo-por-twitter
- Nuestra imagen en YouTube: https://youtu.be/TPFvd0QTxbc

Glosario Estándar de términos utilizados en pruebas de software

SSTQB - Spanish Software Testing Qualification Board

http://www.sstqb.es/ficheros/sstqb_file94-5205ea.pdf

Ofimática y herramientas de colaboración

Veremos distintas herramientas que tienen funcionalidades de colaboración: correo, calendario, tareas, contactos, chat.

Correo

Es el principal medio de comunicación escrita en ambientes empresariales. DEBES TENER UNO.

Elementos importantes:

- **Para**: uno o más destinatarios del correo.
- **CC** (con copia): gente que tiene que estar informada del tema, pero no son decisores.
- **CCO** (con copia oculta): gente que va a recibir el correo, pero no es visible para el resto; ojo con que estos respondan, el resto no sabía que estaban copiados; puede ser una buena opción para comunicaciones masivas (para no divulgar los correos y para evitar el responder a todos).
- **Asunto**: título, breve descripción del motivo del correo (¡¡¡no dejarlo vacío!!!).
- **Cuerpo del correo**: mensaje detallado que se quiere transmitir.
- **Adjuntos**: es posible adjuntar uno o más archivos que sirvan de complemento al mensaje que se quiere transmitir (como las capturas de pantalla en Mantis); tener cuidado con los tamaños de los adjuntos.
- **Responder**: genera un correo únicamente dirigido a quien nos había enviado el mensaje que estamos respondiendo (no se incluyen los adjuntos que contenía el correo original).
- **Responder a todos**: genera un correo dirigido a quien nos había enviado el mensaje que estamos respondiendo y se incluyen todos los que estaban en el campo “Para” y “CC” (no se incluyen los que pudieron haber estado en “CCO”, tampoco se incluyen los adjuntos que contenía el correo original).
- **Reenviar**: genera un nuevo correo con la información del correo original incluyendo los adjuntos; los campos “Para”, “CC” y “CCO” quedan en blanco para que uno ingrese los destinatarios de este nuevo correo.

Organizando el correo en bandejas

Existen diferentes bandejas por defecto donde se organizan los mensajes:

- **Bandeja de entrada**: cualquier correo que recibimos cae en la bandeja de entrada y queda resaltado en negrita indicando que no fue leído.
- **Elementos enviados**: cualquier correo que enviamos va a la bandeja de elementos enviados.
- **Elementos eliminados**: acá van los correos que borramos de cualquier otra bandeja.
- **Borradores**: acá se guardan los correos que estamos escribiendo pero que todavía no enviamos.
- **Archivados/Todos**: es recomendable tener la bandeja de entrada lo más limpia posible, solamente con los correos que no revisamos o que todavía necesitamos tener a mano, entonces todo lo que ya vimos podemos archivarlo para que no moleste.

También podemos crear bandejas para mover los correos que tienen que ver con un determinado tema. Es muy común crear bandejas para cada cliente o cada proyecto, entonces podemos mover a esas bandejas todo lo enviado y recibido para luego encontrarlo más fácilmente.

Nota: en Gmail el concepto de carpetas se hace con etiquetas, entonces uno le pone las etiquetas que necesita a los mensajes y después busca todos los mensajes relacionados con esas etiquetas.

Escribiendo un correo

Saludar como introducción:
- Estimado *Nombre [Apellido],*
- Hola *Nombre,*
- *Nombre,*

Ser concreto. Una breve introducción del motivo del correo, cuál es el problema o mensaje a comunicar, qué necesitamos, etc.
Si es necesario, algún párrafo más con detalles o descripciones del tema.

Párrafo de cierre: qué esperamos, qué conclusión sacamos, etc.
Despedirse:
- Atentamente, *Nombre Apellido (remitente)*
- Saludos, *Nombre [Apellido] (remitente)*

Los correos enormes no se leen: si tienen más de 3 párrafos piensen si no se puede resumir, si tienen que hacer scroll (más de una pantalla) algo está mal.

Antes de enviar el correo:
- LEERLO: a ver si dice todo lo que necesitábamos decir en una forma entendible
- Revisar faltas de ortografía con el corrector de la herramienta que estemos usando

Sugerencias varias:
- NO se escribe TODO EN MAYÚSCULA. Las mayúsculas van sólo al comienzo de las oraciones (después de un punto) y nombres propios.
- Responder entre líneas en otro color (preferentemente no rojo) es una buena técnica cuando en el correo que nos enviaron nos hacen varias preguntas; en estos casos la respuesta es siempre breve, no más de 2 líneas
- Si recibí un correo que necesito ver más tarde con más atención, marcarlo como no leído para no olvidarme
- Correo no deseado (SPAM): no reenviar correos no relacionados con temas laborales: cadenas, ofertas, etc. (está comprobado que no se van a morir si no lo hacen, ni se van a hacer millonarios si lo hacen)
- El correo no sustituye el teléfono

Links de interés (correo)

- Reglas para usar el correo electrónico correctamente: http://es.ccm.net/contents/119-reglas-para-usar-el-correo-electronico-correctamente
- Las 9 reglas básicas de Google para escribir correos: http://www.forbes.es/actualizacion/2194/las-9-reglas-basicas-de-google-para-escribir-correos
- Escribir un correo electrónico – Reglas de etiqueta: http://www.englishcom.com.mx/redaccion/escribir-un-correo-electronico/
- Diez reglas esenciales de etiqueta para el email: http://elemail.about.com/od/consejos/tp/Diez-Reglas-Esenciales-De-Etiqueta-Para-El-Email.htm
- 17 normas de etiqueta para los e-mails: http://www.marketingdirecto.com/actualidad/e-mail-marketing/17-normas-de-etiqueta-para-los-e-mails/

Calendario (Citas)

Las citas tienen:

- Asunto: qué motiva esa instancia.
- Lugar: dónde se va a hacer la reunión, puede ser una dirección, una sala, oficina o un número de teléfono.

- Descripción: donde se puede escribir más en detalle a qué hace referencia la cita. En caso de ser una reunión se usa para poner de qué se va a hablar, qué material se va a compartir o qué deberes tienen que hacer los invitados antes de ir a la reunión. Se pueden incluir adjuntos
- Invitados: ahí pasa a ser una reunión.
- Duración: las citas tienen una hora de comienzo y finalización dentro del mismo día: ej. 1ra reunión cliente x, próximo lunes de 9 a 10.
    - Existen instancias de día completo (por ej. jornadas de integración). En estos casos es más o menos equivalente marcarla con la opción “todo el día” o de 9 a 18hs 8 (salvo por el momento en el que se recibirá el reminder).
- Pueden repetirse (periodicidad): en estos casos se hace una cita con la duración correspondiente y se configura la periodicidad correspondiente.
- Recordatorio: se puede poner una alerta x minutos antes de la hora de comienzo de la reunión (considerar tiempos de traslados).

Ejemplos de citas recurrentes:

- Objetivos para semana, todos los lunes de 9:30 a 10.
- Festejos de cumpleaños del mes: último viernes de cada mes, de 13 a 14hs.
- Jornadas de integración de más de un día: todo el jueves y todo el viernes, se puede hacer de varias formas:
    - desde el jueves hasta el viernes todo el día (una cita sola),
    - desde el jueves a las 9 hasta el viernes a las 18hs (una cita sola),
    - el jueves a las 9 hasta las 18hs con una repetición el día siguiente (esta es la mejor opción porque permite mover una instancia sin afectar la otra,
    - una cita el jueves y otra el viernes (no recomendable porque hay que duplicar la información en la descripción, los destinatarios, etc.).

Las citas se pueden reenviar: si uno es invitado a una reunión y considera que alguien más tiene que ir (que no fue incluído por el que hizo la reunión) se puede reenviar. En estos casos considerar chequear con el organizador si esa persona está bien que participe; también tener en cuenta que el organizador recibe un aviso que la reunión fue reenviada.

Cuando uno recibe una cita tiene diferentes opciones:

- Aceptar: acepta la fecha/hora propuesta
- Rechazar: no acepta.... por los motivos que sea
- Provisional: uno todavía no puede decidir si puede o no puede asistir
- Proponer nuevo día/hora: uno no puede a la fecha/hora propuesta entonces le propone al organizador hacerla en otro momento (el organizador acepta o no ese cambio)

Contactos

Todos los sistemas de colaboración manejan contactos: Outlook, Gmail, etc.
El contacto es como una ficha personal para cada persona con la que interactuamos (un contacto por persona con toda la información relevante).

Es importante completar correctamente todos los campos de los contactos, o al menos los principales:

- Nombre
- Apellido
- Tel/Cel de contacto

- Correo de contacto
- Empresa
- Puesto/Cargo
- Otras cosas que ayudan:
  - foto: a veces se puede sacar de internet
  - fecha de cumpleaños

Completar correctamente la información en los campos correspondientes ayuda a encontrarla rápidamente y a que los programas reconozcan los nombres cuando se los incluye en un correo o reunión. Actualmente las agendas y contactos se sincronizan con los teléfonos, entonces tener la información prolija ayuda a la hora de tener que llamar a alguien o escribir un mail desde el celular.

Skype/Hangouts

Existen varias herramientas de chat integradas a los servicios de correo. Las más usadas son Skype (Microsoft / Office) y Hangouts (Google / GMail).

En los ambientes corporativos los chats se usan para comunicar un mensaje o mantener una breve conversación o intercambio de ideas con alguien. En general se reserva para instancias informales y de las cuales no se necesita mantener un registro para el futuro.

Al igual que en el correo, en conversaciones laborales se recomienda no abusar de los gestos y emoticones (salvo entre compañeros cuando hay confianza).

Estas herramientas también permiten llamadas y videoconferencias, lo que facilita la comunicación en equipos distribuidos geográficamente y reduce costos en llamadas internacionales.

Herramientas on-line / almacenamiento en la nube

Existen herramientas que nos permiten guardar nuestros archivos “en la nube”, o sea, en servidores que se pueden acceder por internet. Esto tiene la ventaja de que podemos acceder a nuestros archivos desde cualquier lugar, quedan respaldados, y podemos verlos desde cualquier PC o dispositivo móvil.
Algunos ejemplos:

- Google Docs
- Office on-line
- Drive
- OneDrive.

Muchos de estos son gratuitos, y sólo si se necesita más funcionalidad o mucho espacio, es donde uno debe pagar.
Te recomendamos que te hagas una cuenta en alguna de esas opciones, y guardes ahí por ejemplo tu Curriculum.

Procesadores de texto (Word / Google Docs)

Sirven para escribir documentos, incluyendo imágenes, diagramas, tablas, etc.

Los cuidados más importantes a la hora de hacer documentos en un procesador de textos es cuidar los formatos.

Usar siempre el corrector ortográfico de la herramienta.

Formatos

Formatos de fuente (tipografía)

Los formatos de fuente refieren al tipo de letra, tamaños, colores y adornos (negrita, subrayado, itálica, etc.).

Cuidados a tener a la hora de hacer documentos:

- elegir tipos de letra fáciles de leer: arial, calibri, etc.
- cuidar los colores, usar preferentemente negro o azul marino como máximo.

Formatos de párrafo

Un párrafo es todo el texto que existe antes de un salto de línea (enter).
Las características más importantes para considerar en relación a los párrafos con:

- **alineación**: izquierda, centrado, derecha, justificado (va de margen a margen),
- **espaciado**: espacio antes y después de cada párrafo,
- **interlineado**: espacio entre líneas.

Títulos y estilos

Para poner títulos es conveniente usar los estilos que vienen predefinidos en lugar de ajustar los tamaños a mano. Esto hace que todos los títulos queden iguales y facilita la creación de índices.

Tablas

Cómo insertar una tabla. Las tablas de Word permiten de una forma rápida y simple estructurar, organizar o ajustar textos dentro de un documento Son ideales para acomodar listados con múltiples campos tanto de texto como numéricos. También pueden servir para agrupar párrafos junto con otros párrafos o gráficos.

Archivos PDF

Es un formato no editable. Para poder visualizarlos usas el navegador, o un programa como por ejemplo Adobe Acrobat Reader. ¿Lo tienes instalado?
Sirve para compartir documentos con clientes o cualquier persona externa donde importe que el otro no cambie el documento.

¿Qué comunica nuestro curriculum?

¿Has pensado sobre qué comunica el CV? ¿Qué cosas queremos que se transmitan?

Por ejemplo: responsabilidad, que somos ordenados, que le ponemos ganas, que queremos trabajar.

¿Cómo lograr que el CV muestre esas cosas?, y ¿cómo nosotros al enviarlo y presentarlo tenemos que actuar para lograr eso mismo?

Cuando hablamos de comunicación nos referimos a conversaciones y también a comunicación escrita y a todas las cosas que decimos, no importa en qué formato. Por ejemplo, cuando enviamos un curriculum a alguien queremos decirle algo, ¿no? Algo así como “considerame para este trabajo”.

Si queremos decirle a alguien que nos considere para un trabajo X, ¿qué te parece que deberíamos decirle?

- que tenemos ganas,
- que tenemos la capacitación necesaria (o estamos dispuestos a aprender),
- que somos responsables y tenemos las cualidades necesarias para el trabajo.

¿Cómo te parece que podemos transmitir estas cosas en el curriculum?

Ganas: entregando un currículum llamativo/interesante y hacerlo en tiempo y forma.

Capacitacion: listando nuestras calificaciones de forma correcta (lo más relevante arriba).

Cualidades: que somos prolijos, ordenados, creativos, en fin, lo que gusten señores.

¿Qué tiene un buen CV?:

- Datos personales
- Objetivo profesional
- Experiencia laboral
- Estudios
- Otros estudios
- Referencias laborales
- Referencias personales

Si quieres leer más sobre curriculum:

- http://www.modelocurriculum.net/reglas-basicas-para-escribir-el-curriculum.html
- http://www.mejorartucv.com/10-basicos-del-curriculum-vitae/

Si quieres leer más sobre comunicación:

- http://testingbaires.com/la-importancia-de-la-comunicacion-interpersonal-para-un-tester/
- http://blog.ces.com.uy/?p=159
- http://www.softqanetwork.com/la-importancia-de-las-soft-skills-en-el-testing
- https://josepablosarco.wordpress.com/2011/01/06/%C2%BFporque-realizar-testing/

Presentaciones Orales

Aquí te dejamos algunos videos de Sebastián Lora, los cuales muestran puntos importantes para la comunicación oral:

- La primera clave de la persuasión al hablar en público (1:24): https://www.youtube.com/watch?v=sJ0zb5jBVDE
- El mensaje a transmitir (1:38): https://www.youtube.com/watch?v=y_qQn80feXg
- La planificacion - el objetivo (1:28): https://www.youtube.com/watch?v=ivg2R2AGfEs
- El flujo de la ansiedad (3:31): https://www.youtube.com/watch?v=tG9HOqFvBJU
- Menos es más (1:26): https://www.youtube.com/watch?v=6XlD5bQcGVM

Resumen de los videos

1) Tenemos un minuto para decirle al público por qué nos tiene que prestar atención, qué va a ganar, en qué se beneficiará atendiendo a nuestra presentación.

2) Planificación:

a. Tenemos que saber cuál es nuestro público.

b. Objetivo: qué queremos lograr con la presentación.

c. Finalidad: cuando haya terminado la presentación, qué quiero que el público haga, piense o sienta.

3) El mensaje: qué es lo único que se tienen que acordar, la idea fundamental

a. Debe ser corto: lo bueno y breve, 2 veces bueno.

b. Concreto: escrito en una frase que el público pueda entender.

c. Beneficio relevante para el público específico, debe contestar a la pregunta ¿por qué debería interesarme?

4) Captar la atención rápidamente: El ser humano tiene una capacidad de atención de 10 segundos, así que hay que sorprenderlo antes que eso. El desafío es captar la atención del público rápidamente; puede ser con una pregunta interesante, una historia breve, cualquier cosa que haga que la gente no diga "esto es más de lo mismo" y se vaya a mirar el celular.

5) El stress

a. Cuando nos enteramos que tenemos que hacer una presentación, el stress hace que nos olvidemos del tema hasta poco tiempo antes de hacer la presentación, cuando ya no nos va a dar el tiempo para prepararla bien. Lo ideal es empezar a preparar la presentación con tiempo suficiente.

b. El último momento (el día antes): si la presentación ya está lista, conviene ubicarse en un lugar tranquilo y repasar mentalmente la presentación, contando el tiempo y sintiendo confianza en que va a salir bien.

c. Buscar caras amigables en el público, gente que sonríe, para contagiarnos de esa alegría y no sentir que estamos siendo juzgados.

Perfil digital

Redes sociales

Es muy importante hacer buen uso de las redes sociales. Esto lo veremos con algunos ejemplos que nos ayudan a reflexionar juntos.

Debemos ser conscientes de que todos emitimos juicio de valor al ver una foto (o comentarios) en las redes sociales, y que el objetivo principal es al menos **llegar a la entrevista**. Todo el mundo está en internet, y cuando alguien piense en tener una entrevista con nosotros, buscará nuestro nombre en Google, Facebook, Twitter y más, y lo importante es que la primera impresión que se lleve no sea negativa.

Ejemplos de personas que tuvieron problemas por este tipo de cosas:

- http://www.instablog9ja.com/wp-content/uploads/2015/02/20150210-143248.jpg

Si bien está en inglés, básicamente la chica emite por twitter un comentario diciendo “mañana comienzo este trabajo de porquería”, a lo que obtiene una respuesta de su empleador diciendo “no te preocupes, mañana no arrancás, estás despedida, que tengas suerte en tu vida sin trabajo y sin dinero”.

- http://www.telecinco.es/informativos/internacional/Kaitlyn_Walls-despedida-Facebook_0_1979250092.html

Los siguientes ejemplos son de dos personas con un perfil de Facebook que puede llevar al empleador a tener prejuicios. Debemos tener cuidado con las fotos y cosas que dejamos públicas y accesibles a cualquiera:

- https://goo.gl/BWoiBi
- https://goo.gl/TdPjE1

Redes sociales: el nuevo jugador en la búsqueda laboral:
http://www.elobservador.com.uy/redes-sociales-el-nuevo-jugador-la-busqueda-laboral-n222491

Fue despedida antes de su primer día de trabajo por Twitter:
http://www.subrayado.com.uy/Site/noticia/41820/fue-despedida-antes-de-su-primer-dia-de-trabajo-por-twitter

Nuestra imágen el YouTube: https://youtu.be/TPFvd0QTxbc

Prepararnos para la entrevista

Repaso del CV

Antes de enviar nuestro currículum, deberíamos repasarlo y asegurarnos que todo está actualizado, y bien ajustado para la propuesta laboral a la cual nos estamos presentando. Les recomendamos repasar lo visto en los primeros encuentros, donde hablamos de cómo armar un cv. Dejamos acá una lista de algunas cosas que ya habíamos visto que son importantes:

- Objetivo / Por qué deberían llamarme.
- Datos personales.
- Datos académicos o Estudio en orden cronológico inverso (lo más reciente más arriba). Poner cosas sobre el curso de Nahual, indicando qué aprendieron en el mismo.
- Otros estudios (idiomas, computación etc.).
- Experiencia Laboral en orden cronológico inverso. Es importante incluir información sobre qué tareas desarrollabas.
- Referencias Laborales. Nombre, empresa, cargo y cel de contacto.
- Referencias Personales. (No incluir padres, tíos, hermanos en esta sección).
- Foto adecuada: recorte en los hombros, actual, vestimenta prolija.

Es importante avisarle a las referencias para que no los tomen de sorpresa si los llaman.

Dónde y Cómo Buscar Trabajo

Existen diversas formas de buscar trabajo, y en este rubro tal vez de las más importantes o las más comunes son las diferentes páginas web y medios digitales.

Veamos algunos ejemplos:

- Buscojobs, Computrabajo, El Gallito, etc.
- Hay redes sociales específicas para buscar trabajo: Linkedin. En esta red social uno arma su perfil cargando información de su curriculum. En lugar de indicar tus gustos, películas favoritas y cuadro de fútbol, como en Facebook, en esta vamos a indicar nuestros conocimientos, estudios, experiencias laborales, etc. También vamos a conectarnos con personas con las que hemos trabajado, o con empresas que buscan y ofrecen trabajo.

Algunas cosas que te recomendamos que hagas al momento de buscar trabajo:

- Leer bien los avisos a los que te presentas, y que estos se adapten a nuestro perfil. Asegúrate que el horario te sirve, el lugar donde se encuentra la oficina, etc.
- Podrías armar una planilla para organizar la búsqueda: Nombre de la empresa, fecha que se envío el CV, cargo al que se postuló, fecha de entrevista. Esto te servirá si te presentas a muchas propuestas y luego te llaman, entonces tendrás información de lo que enviaste, lo que ofrecían, y no te confundirás entre distintas oportunidades. No te olvides que puede pasar que te presentes hoy y te llamen en 3 meses, ¿te acordarás de todos los detalles en 3 meses?
- Si te postulas enviando un mail, nunca dejes el cuerpo del mail vacío. Saluda cordialmente, y luego escribe algunas líneas indicando tus objetivos y tu motivo de presentarte en esa oferta laboral, por qué quieres trabajar ahí.

Cómo me preparo para la entrevista

- Algunas cosas importantes previo a la entrevista:
- Desde el primer acercamiento con la llamada debemos estar preparados. Escuchar bien lo que se nos dice, anotar la dirección, fecha, hora y preguntar el nombre de la persona con la que nos entrevistaremos.
- Investigar todo lo que se pueda de la empresa. En internet hay mucha información, tanto en la página de la empresa, como en Linkedin.
- En la entrevista nos pueden hacer preguntas muy variadas, por ejemplo ¿cuáles son tus fortalezas? ¿cuáles son tus debilidades? No sería bueno que las pensemos por primera vez en ese momento, con lo cual es fundamental conocer nuestras fortalezas y oportunidades de mejora (debilidades), y las expliquemos ahí con tranquilidad. Sería bueno plantear nuestras debilidades con franqueza, y contando que las trabajamos de alguna forma. Por ejemplo, "mi debilidad es que me cuesta hablar en público, por eso intento hablar cada vez que tengo oportunidad, para intentar mejorar".
- Conocer muy bien nuestro CV. Tener bien claro las fechas de inicio y fin. Esto es importante porque nos irán haciendo preguntas sobre lo que está escrito en el CV y no quedaría bien si titubeamos al respecto.
- Vestimenta formal. Lo ideal es investigar cuáles son los códigos de vestimenta de la empresa (buscando en internet, si tenemos un amigo o conocido que trabaja ahí). De otra forma, ir con algo estándar, mínimo con jeans y camisa (por dentro), zapatos o championes simples (sin resortes, sin colores extravagantes).
- Llegar 5 minutos antes. Está muy mal visto llegar tarde, así como también llegar media hora antes (muestra ansiedad). Si llegamos mucho antes es recomendable esperar dando vueltas por ahí.

Expectativas salariales

Esta es una pregunta que en algún momento te harán: ¿cuánto esperas ganar? Lo ideal es no dar un número que no tenga sentido, sino uno en base a lo que se paga en la empresa, y a lo que uno puede llegar a percibir por los conocimientos y el rol que uno puede llegar a ocupar.

La buena noticia es que el sueldo mínimo en esta profesión es bastante alto en comparación a otras ramas, con lo cual pidiendo eso como base ya podremos comenzar ganando bien. Luego a medida que avancemos y mejoremos nuestras habilidades, podremos ajustar nuestro sueldo y así podremos avanzar económicamente.

Los sueldos **nominales** mínimos a la fecha (2015) en el rubro de informática son:

- salario mínimo para técnicos: $22.767.
- salario mínimo para juniors en entrenamiento (máx 9 meses): $14.743.

Esto significa que nuestro salario mínimo es de casi 23 mil pesos, pero que los primeros nueve meses estaríamos a prueba con un sueldo de casi 15 mil pesos.

Para que tengas una idea, 15 mil pesos nominales son aproximadamente 12 mil pesos líquidos.

Durante la entrevista

Actitud

- Trata de mantener una actitud natural, cordial y equilibrada.
- Tienes que mostrarte simpático y atento a la charla.
- No mostrarse ni agresivo, ni sensible, ni arrogante.
- Escuchar con atención las preguntas.
- Piensa bien antes de contestar y responde positivamente.
- Deja que el entrevistador tome la iniciativa y guíe la entrevista.
- Trata de mantener una actitud segura.

Comunicación no verbal

- Siéntate derecho/a, ni al borde de la silla (inseguridad), ni despatarrado (falta de respeto o desinterés).
- Tratá de controlar tus nervios (ojo con los gestos, movimientos, actitudes).

- Ten cuidado con tu modo de sentarte, manos, codos, brazos, taparse la boca.
- Mira al entrevistador a los ojos, pero sin intimidar.

Comunicación verbal

- Saluda al entrevistador cordialmente. Esperar y ver cómo saluda el entrevistador primero.
- En general trata de ser formal en tu comunicación. Intenta modular al hablar.
- No tutees si no te lo indican.
- No interrumpas.
- No hables demasiado, ni demasiado poco.
- No respondas con evasivas o dudas, ni con monosílabos.
- No hables mal de las empresas, jefes y compañeros con los que has trabajado.
- No te niegues a responder preguntas.
- Responde clara y brevemente.
- Di siempre la verdad.
- Agradece la oportunidad de haber tenido la entrevista.

Evaluación Psicotécnica

- Generalmente son preguntas, o dibujar algo, todo en manos de un psicólogo, o con material provisto por uno.
- También en ocasiones hay "juegos" que permiten hacer un análisis de nuestra forma de resolver problemas lógicos. Si quieres practicar puedes hacerlo en estas páginas por ejemplo:
    - http://www.psicotecnicostest.com/
    - http://www.hacertest.com/psicotecnicos/examenes/
    - http://www.psicoactiva.com/psicotecnicos.htm
- Hagan lo que realmente les nazca en el momento.
- No busquen cosas en internet.
- El sicólogo generalmente se da cuenta.

Evaluación Técnica (en testing)

Muchas veces se realizan también pruebas técnicas de testing. Esto puede incluir que les pidan que hagan pruebas de diversas cosas, como, por ejemplo:

- Aplicar una técnica para probar un sistema (pueden aplicar clases de equivalencia, valores límites, y si se animan, árboles de decisión)
- En ocasiones les pueden preguntar cómo probarían algo menos convencional, por ejemplo, una máquina expendedora de café. En estos casos es importante marcar que:
    - Deben interactuar con la máquina.
    - Deben probar diversos valores y ver su resultado. Por ejemplo, seleccionar sin azúcar y verificar que no le ponga azúcar.
    - Luego, probar con valores inválidos, qué pasa si no hay café, si no está enchufada, si no hay agua, si presiono todos los botones a la vez, si no pongo plata, etc.
    - Ojo con indicar que van a usar una técnica, pues si indican que van a aplicar árboles de decisión, ¡luego lo tendrán que hacer muy bien!

¿Y después de Nahual qué...?

Formación

Aquí se le muestran diferentes opciones para seguir ampliando sus estudios en el ámbito informático como la:

- **Presenciales en Uruguay:**
    - **UTU**: http://www.utu.edu.uy/ Existe un bachillerato en informática.
    - **UTEC**: http://www.utec.edu.uy/es/ (TECNICATURA EN TECNOLOGÍAS DE LA INFORMACIÓN).
    - **INEFOP:** http://www.inefop.org.uy. El Programa está dirigido a jóvenes de ambos sexos, entre 15 y 29 años, con necesidades de incrementar sus condiciones de empleabilidad.

- **Cursos Online (por internet, e-learning)**
    - **CES**: Privado, con costo. La primera parte es abierta, con lo cual se puede ver gratuitamente. http://capacitacion.ces.com.uy/. Convenio con INEFOP (Instituto Nacional de Empleo y Formación Profesional) las personas que hayan terminado el liceo y se encuentren sin empleo se les otorga media beca. La aprobación de la beca está sujeta a una entrevista previa realizada por el CES e INEFOP.
    - **Cursos gratuitos**: existen cientos de cursos gratuitos de todo tipo, desde herramientas de ofimática (word, excel, etc), operador Windows, testing, programación, y muchísimo más. Sólo necesitas tiempo, ganas, voluntad, internet y una computadora. Te pasamos algunas opciones, pero hay muchas más:
        - Ofimática
            - http://cursosgratuitos.eu/informatica-diseno-y-programacion/ofimatica/
            - http://www.deseoaprender.com/
            - http://www.aulafacil.com/cursos/c68/informatica/excel-word-powerpoint-access
            - http://www.aulaclic.es/index.htm
        - Testing
            - http://www.softqanetwork.com/el-inteco-ofrece-cursos-gratuitos-especializados-en-calidad-de-software
        - Programación
            - https://www.yopuedoprogramar.com/
            - http://program.ar/
            - http://www.formaciononlinegratis.net/
            - http://training.genexus.com/principal/inicio?es
        - Aprender Ingles
            - http://www.aulatutorial.com/cursos/idiomas
            - http://www.educatina.com/ingles
            - http://duolingo.com/es
            - http://livemocha.com/
            - https://www.verbling.com/
            - http://colingo.com/
            - https://www.youtube.com/user/TeacherPhilEnglish
            - http://www.engvid.com/
            - http://www.englishcentral.com/videos
            - http://www.subingles.com/
            - http://www.italki.com/
            - http://www.abaenglish.com/en/

Convertirte en colaborador Nahual

Estás más que invitado a pasar al otro lado del mostrador, a convertirte en uno más del equipo de colaboradores. Es una forma de seguir aprendiendo, y lo más importante, compartir lo que aprendiste con otros, con los que vendrán en los próximos ciclos. De esta forma podrás seguir acercándote y vinculándote con más gente que está en el rubro, y que lo estará posiblemente a futuro.

Animáte y contáctate con nosotros para ver cómo podés ayudar y seguir en sintonía con el equipo. Podés escribirnos a nuestro e-mail (sumate@nahual.com) o directamente a través del grupo de egresados en Facebook (https://www.facebook.com/groups/289151604559114/).

PROYECTO NAHUAL ARGENTINA