DE JUNIOR AL INFINITO

Una guía para vivir, sobrevivir y disfrutar el apasionante viaje del desarrollo de software

ENZO TRUCCHI

Trucchi, Enzo
De junior al infinito : una guía para vivir, sobrevivir y disfrutar el apasionante viaje del desarrollo de software / Enzo Trucchi. - 1a ed - Villa María : Enzo Trucchi, 2021.
Libro digital, Amazon Kindle

Archivo Digital: descarga
ISBN 978-987-86-8419-2

1. Aplicaciones Informáticas. 2. Autoaprendizaje. 3. Autoayuda. I. Título.
CDD 158.1

A mi familia. Son tantas y tantos que te aburrirías.

Contenido

Contenido 4
INTRODUCCIÓN 5
QUIEN SOY 7
COMO LEER EL LIBRO 9
Capítulo 1 ATERRIZAJE 10
¿POR QUÉ ES TAN INTERESANTE EL MUNDO DEL DESARROLLO? 10
ENCAJE PERSONA EMPRESA 16
STRESS/BURN OUT 19
ENTREVISTAS 22
TECNOLOGÍA COMO IMPULSOR DEL CAMBIO 24
Capítulo 2 HABILIDADES BLANDAS 27
QUE ESPERAN DE VOS 27
ESTIMACIÓN 30
HÁBITOS 33
Capítulo 3 HABILIDADES TÉCNICAS 35
HERRAMIENTAS 35
LENGUAJES 38
ESTÁNDARES Y BUENAS PRÁCTICAS 42
APRENDÉR A APRENDÉR (y a buscar) 58
RECURSOS 59
PALABRAS FINALES 60
ABRAZAR EL CAMBIO 60
NO TE COMPARES 61
MEMENTO MORI 61

INTRODUCCIÓN

¿Qué le dirías a una persona que está arrancando para que sea mejor dev?
¿Cómo invitarías a alguien a que se sume a esta profesión?

Nadie me contó cómo era realmente este mundo. Y la poca claridad que se ve desde afuera es un gran problema a la hora de invitar a que más personas se unan e intenten recorrer el camino. Si nos detenemos a pensar, pasa en otras carreras también y no sólo en el plano que involucra la parte profesional de nuestras vidas, sino que es un factor en común en muchos aspectos. A veces lo único que vemos adelante es neblina y nos es difícil tomar una decisión trascendental. Como plus, no sabemos que puertas tocar. Más que nuestro núcleo familiar y alguna persona que conozcamos, la decisión es nuestra y en muchas ocasiones no estamos preparados para tomarla.

Hablando de neblina y lo desconocido. Sentimientos de soledad, aislamiento, falta de resultados o inclusive síndrome del impostor [1]son moneda corriente cuando decidimos dar un paso tan importante como comenzar una nueva carrera o un nuevo trabajo. Por lo general al momento de elegir estudiar algo, aún tenemos dudas, aún no sabemos con qué vamos a encontrarnos y no conocemos mucha gente. Tenemos por delante un cambio de ambiente en donde estábamos cómodos (o tal vez no, pero lo conocíamos), convivíamos a diario y podíamos tener argumentos suficientes para determinar si queríamos seguir perteneciendo a esa comunidad.

Este libro es una búsqueda con el fin de minimizar toda esa neblina y hacer tu camino lo más claro posible.

Cuando estaba aún decidiendo qué hacer con mis pocos aprendizajes y que rumbo encarar profesionalmente, me hubiese gustado tener más certezas de lo que significaba ser parte de una revolución tecnológica, de una empresa de sistemas o de un área especializada en tecnología y en qué debería haber hecho foco para ser un mejor jugador de equipo. Probablemente hubiese buscado de forma más temprana empresas en donde desarrollarme que se asemejaran más a lo que yo quería hacer y a mis ideales. Tampoco es que tengamos muy en claro que queremos hacer en etapas iniciales de nuestro recorrido profesional.

En mi caso la decisión de incursionar en el mundo tecnológico vino después de hacerlo en la contabilidad, luego de seis meses y la decisión final de que esa carrera no era para mí. Al margen de eso, no tuve ningún tipo de revelación ni de información que apoyara esta segunda decisión (y potencial error). Podría haber elegido cocina, administración rural o abogacía. En ningún caso tenía información de valor que pudiera aportarle luz a esa encrucijada.

Por otro lado, y en otro contexto diferente al mío, también podría suceder que no tengas la posibilidad de equivocarte y seguir intentando, o podría ser que no te animes a dar el paso a dejar una carrera, logros conseguidos, una buena remuneración y muchos beneficios más. En ese caso, y si te interesa conocer que hace falta para ser parte de este mundo, este libro podría servirte.

Antes de contarte un poco sobre mí, quiero compartirte una situación muy particular:

Con el paso del tiempo, conocí a muchas personas que necesitan un empujón para tomar la decisión final sobre incursionar en este mundo, y cada vez que tengo la posibilidad de hablar con estas personas, veo que es un anhelo cada vez más lejano. Es un tema de perspectiva a mi entender, ya que aún mucha gente cree que los que hacemos código somos mutantes. Decidí entonces escribir un libro porque soy de las personas que quieren ayudar de alguna forma a que el futuro en tecnología no sea sólo una utopía, considero que tengo las herramientas necesarias para hacerlo, y creo que si puedo mostrarte aunque sea un pequeño pantallazo de lo que es este maravilloso mundo, vas a querer ser parte inclusive antes de terminar de leer el último capítulo.

Entonces, la idea es clara: ayudar a que las personas que decidan incursionar en desarrollo tengan una base o un pantallazo que les permita definir si siguen o no este camino. Y si lo siguen, que sea lo más llevadero posible. No importa si estás empezando a estudiar después del secundario o si cambiaste de

[1] El síndrome del impostor, a veces llamado fenómeno del impostor y acuñado como fenómeno del impostor, es un fenómeno psicológico en el que la gente se siente incapaz de internalizar sus logros y sufre un miedo persistente de ser descubierto como un fraude. A pesar de las pruebas externas de su competencia, aquellos con el síndrome permanecen convencidos de que son un fraude y no merecen el éxito que han conseguido. Las pruebas de éxito son rechazadas como pura suerte, coincidencia o como el resultado de hacer pensar a otros que son más inteligentes y competentes de lo que ellos creen ser. https://es.wikipedia.org/wiki/S%C3%ADndrome_del_impostor

profesión porque querías probar en sistemas. Este libro va a servirte en esa etapa inicial: va a ayudarte en la preparación y aterrizaje al mundo maravilloso de IT, y va a permitirte escalar de una forma sana a través de las diferentes montañas del desarrollo.

En ese caso, misión cumplida.

QUIÉN SOY

Mi nombre es Enzo y soy un apasionado por la tecnología. Mis primeros pasos en este maravilloso mundo fueron siendo un niño de 6 años, rompiendo la computadora familiar, en la que teníamos juegos old school y algún que otro programa que corría en ese flamante Windows 95.

Pero adelantémonos un poco. Mi familia tuvo un negocio gastronómico por años, por lo que el trabajo siempre fue algo natural. Además de aprendér habilidades relacionadas a la responsabilidad y a tener un balance entre nuestra vida – trabajo, tener la posibilidad de trabajar en un entorno familiar en donde primaba el respeto y poder empezar a probar y cometer equivocaciones desde una edad temprana abre muchas puertas en el futuro. Te enseña a reaccionar y a hacer frente a situaciones estresantes y traumáticas en el entorno laboral. Calculo que haber tirado algunas bandejas al piso con 15 años y un salón lleno suele ser motivo de vergüenza al principio, pero eso puede haberme preparado para reaccionar de forma prolija y proactiva después de “tumbar” una que otra base de datos de producción (el que esté libre de pecados que tire la primera piedra).

A los 19, después de haber dejado la carrera de Contador Público con 6 meses de cursado y sin dejar mis responsabilidades en el negocio de la familia, me decidí por algo que iba por una vereda totalmente distinta y me inscribí en el Instituto Leibnitz de Villa María para cursar Analista de Sistemas de Computación, como expliqué antes, sin muchos datos que apoyaran esa decisión en contra de seguir una carrera orientada a cualquier-cosa-que-se-te-ocurra y se abrió un mundo nuevo de posibilidades. Tenía conocimientos nuevos, un poco básicos pero conocimientos al fin. Además, estas nuevas habilidades técnicas no eran fáciles de adquirir (no había un boom de sitios o apps para ver tutoriales y hacer cursos). Inmediatamente quise empezar a hacer algo al respecto y empecé a buscar en dónde probar lo que estaba aprendiendo. Me contacté con algunos profes pero siempre resultaba que me faltaba algo más que aprendér para llevar a cabo una idea. Ganas, por otro lado, sobraban.

En el segundo año de la carrera arranqué a trabajar un poco de forma informal, con las horas que podía, con la misma persona que nos había arreglado esa máquina que había dejado sin uso (todo sucede por una razón, claro que sí). No tenía horarios fijos ni tareas asignadas, la idea era hacer un upgrade del software que manejaban y yo entraba en escena como super junior a aprendér tecnologías nuevas, agregar funcionalidades, tratar con clientes, manejar bases de datos y algunas cosas más. No estaba mal y siempre voy a estar agradecido a esa persona que me permitió empezar a hacer armas en este mundo, pero yo necesitaba un poco más de estabilidad. Ya tenía 20 y planes de irme a vivir solo y viajar por el mundo.

Por cuestiones del destino (o de un mal seguimiento de parte de la empresa que nos lo vendió), cuando cursaba tercer y último año de la carrera, mis viejos decidieron que el sistema que habíamos adquirido en el bar para llevar las cuentas corrientes de clientes habituales ya no iba para más. Ahí entraba yo, ofreciéndome para, además de hacer las horas que ya tenía atrás de la barra, desarrollar el sistema con lo que estaba aprendiendo. Total, tenía que rendir Programación II y me venía al pelo para ponerme al día con los conceptos y tipear un poco de código.

En unos meses salió andando y fue un gran reemplazo. De paso, aprobé la materia (con un poco de código reutilizado, claro). Ese sistema de cuentas corrientes fue mi primera experiencia real desarrollando algo propio y un proyecto en el que trabajé a conciencia durante noches para lograr un resultado del que estuviera orgulloso y que fuera funcional, sintiendo la presión de querer entregar a tiempo según lo acordado. No conocía el término MVP ni mucho menos, pero me imagino que el entregable fue muy parecido a eso.

El juego estaba comenzando.

Cuando al año siguiente mi familia decidió vender el negocio que había sido nuestro sustento por tantos años (ey, que el sistema no tuvo nada que ver), intenté hacer un microemprendimiento tirando a cuestiones técnicas, arreglando máquinas y cosas por el estilo. En ese momento no había llegado el recambio de notebooks -aunque muchos hacía un par de años que las veníamos utilizando- y las computadoras seguían siendo de escritorio, en muchos casos con muchos más años de lo recomendado para utilizarlas, por lo que había buena variedad. Yo mucho de cuestiones técnicas no entendía, pero contactos tenía y ganas también. Hice unas tarjetitas y unos stickers para pegar en los equipos que iba a arreglar, y listo, negocio en marcha. O no. Debe haber durado 4 meses en total, con 3 máquinas arregladas con éxito y 1 que nunca más volvió a respirar. Esa última fue la que apuró a que cerrara el pequeño taller/emprendimiento.

Después de eso estuve un tiempo en la Municipalidad de Villa María como soporte de un sistema orientado a la Salud, específicamente a digitalizar historias clínicas y facilitar el acceso a la información de

salud. En esa etapa de 2 años casi no vi código. Estaba más orientado a solucionar problemas de la diaria de usuarios y usuarias a las que les costaba usar el sistema. Fue una linda experiencia pero sentía que recién estaba conociendo la punta del iceberg respecto al mundo del desarrollo. Además, no es lo mismo trabajar en un ámbito público que en uno privado, eso es seguro.

Con el título en la mano y con ganas de seguir conociendo y haciendo, no iba a durar mucho. Al poco tiempo estaba tocando puertas de vuelta, buscando avisos en el diario (sí, antes era normal). Buena decisión. Me llamaron de una empresa local de desarrollo e implementación de software, en la que estuve dos años tocando un poco de código y aprendiendo a ser parte de un equipo más grande que el anterior, cosa nueva para mí.

Al mismo tiempo, vendíamos nuestro primer software al Área de Salud de mi anterior empleo. Tuve la suerte de coincidir con una persona pionera en Informática de Salud y con ganas de hacer cambios reales, y eso derivó en que armara un equipo de 2 desarrolladores y nos pusiéramos a trabajar codo a codo para sacar una solución orientada a la salud cardiovascular. Con esa persona hoy en día seguimos intentando hacer cosas geniales apoyándonos en la tecnología, buscando cambios reales y significativos en cuestiones de salud.

Académicamente, ya me había recibido de Analista de Sistemas, un título de 3 años, y estaba cursando la Licenciatura en Informática. Seguía buscando mi camino, seguía pensando que se podían hacer cosas geniales en el mundo de la informática teniendo el lugar indicando en donde desarrollarse. Claramente no era donde estaba en ese momento. Y no porque no me hubiesen dado la posibilidad. Simplemente no era el lugar. Más tarde voy a contarte un poco sobre el encaje persona/empresa y cómo hacer para ir teniendo pequeños descubrimientos que nos indiquen que estamos en el lugar indicado o no.

Cuando decidí sumarme al barco (¿o al camión?) de la empresa en la que estoy actualmente, Pedrolga, supe que ese era el lugar en donde iba a desarrollarme. Supe que iba a defender esa relación e intentar que sea ganar-ganar con honestidad y trabajo duro, siempre. Hoy en día, pasaron 4 años y un poco más desde que defiendo esos colores. Cambié de puestos algunas veces, siempre relacionado a la tecnología, innovación y creatividad, y el camino que queda por recorrer es inmenso, pero lo que crecimos juntos es incalculable.

CÓMO LEER EL LIBRO

Este libro está dividido en cuatro capítulos, cada uno con secciones cortas pero que tratan de forma precisa los temas que quiero exponer. Si no sos una persona que lee con regularidad, te recomiendo que lo trates con calma y disfrutes cada capítulo en particular. Podés leerlo las veces que sea necesario, tratar de entender las cuestiones técnicas (también revalidarlas en línea), y sacar tus propias conclusiones. Si encontrás algo que te haga un poco de ruido, mi mail está disponible para revisarlo:

trucchienzo@gmail.com

El detalle de los capítulos es el siguiente:

- Capítulo Uno. Aterrizaje: Un aterrizaje no forzado en donde vamos a ver al mundo del desarrollo desde diferentes ópticas como la psicológica, la comparación con otros caminos que podrías tomar y viendo a la tecnología como un motor de cambio cultural.

- Capítulo Dos. Habilidades Blandas: Habla del camino en el mundo del desarrollo desde un punto de vista de habilidades blandas, teniendo en cuenta que al ser junior (y de ahí en adelante), no sólo vas a tener que desarrollarte en cuestiones técnicas.

- Capítulo Tres. Habilidades Técnicas: Vamos a ver la parte técnica del desarrollo. Teniendo en cuenta que tal vez conozcas poco de definiciones o de herramientas, es un capítulo corto y en el que traté de resumir lo que creo que sería bueno que sepas sin necesidad de tener experiencia laboral.

- Palabras finales: Algunas conclusiones que creo relevantes para el camino que estás por emprender.

Voy a tratar de tocar los temas esenciales con los que te vas a topar si decidís incursionar en la tecnología (o si ya lo hiciste pero aún estás en la búsqueda de ese lugar a donde puedas desarrollarte y creés, como yo, que el conocimiento siempre suma).

Capítulo 1
ATERRIZAJE

¿POR QUÉ ES TAN INTERESANTE EL MUNDO DEL DESARROLLO?

Cuando hablamos de desarrollo de software, es muy difícil explicar a las personas fuera de ese círculo que coexisten muchas especializaciones y que no todas las personas podemos hacer lo mismo o cumplir los mismos roles en equipos de trabajo. Imaginate una situación en donde querés contarle a alguien cercano a que querés dedicarte (o en qué te querés especializar) dentro de la esfera del desarrollo:

- Sí, quiero ser Front End dev.
- ¿Te conté? Estoy cursando una especialización en DevOps.
- ¡Enhorabuena! En mi equipo ascendí a Ssr. Back End Dev con orientación a Node Js.

Probablemente la persona que reciba tu mensaje esté pensando en alguien sentado frente a algunos monitores negros con letras verdes, al mejor estilo Matrix.

Vamos de a poco. Hoy es inmensa la cantidad de especialidades en las que podés trabajar, todas dentro del mismo paraguas. Esto es algo que personalmente no tenía claro cuando empecé, y lo menciono al principio porque creo que es uno de los grandes diferenciales de esta profesión.

¿No estás seguro si desarrollar en tal tecnología va a convencerte? Ok, es fácil cambiar.

¿No estás convencida de querer hacer el tipo de trabajo técnico para el cual te estás formando? Ok, se puede ir por otro lado, siempre perteneciendo al mundo del desarrollo.

¿Las herramientas que la empresa te provee no te convencen? Genial, podés investigar y proponer otras.

Entonces, aclaremos algunas cosas:

¿Somos tan especiales los que nos dedicamos a esto?

¿Hay algún tipo de magia oculta o de visión o es simplemente trabajo duro, a conciencia, con paciencia y como en casi cualquier otra disciplina?

Empiezo el libro de esta manera, utilizando una óptica ajena a la nuestra e intentando descifrar cómo nos ven desde afuera, por un solo motivo:

Si vos estás empezando a transitar este camino, es probable que también lo veas así. No dudo que hayas hecho la misma cara de incredulidad cuando leíste en algún lado que se necesitaban personas con Seniorityes elevados para cubrir tal vacante, o que la tecnología que habías escuchado alguna vez había sacado su versión número 9 en el último año, promediando 2 beta anuales. La idea es que lo encares de a poco, teniendo en cuenta que este mundo es muy profundo y podrías marearte rápido. Por eso, quiero que sepas que el punto de partida generalmente es uno solo pero luego, y con mucho esfuerzo, el camino empieza a abrirse y nos encontramos con muchísimas alternativas diferentes para crecer, desarrollarnos y ser felices.

No te olvides de eso. Ser feliz.

Ser desarrollador o desarrolladora, sea cual sea tu posición, jerarquía o forma de llevarlo a cabo, es algo grandioso. A su vez, mientras más grande sea el proyecto en el que puedas participar, más posibilidades tenés de participar con otras personas con especialidades diferentes. Devs Back End, Front End, devops, administradores de base de datos, analistas funcionales, testers, qa testers, scrum masters, ingenieros de seguridad, líderes de proyectos y la lista es interminable -de verdad-. Vos también podés seguir alguno de estos caminos: intentar especializarte en una *skill* específica y tratar de armar tu carrera en base a esa (seguramente necesites conocimientos en algunas otras), o ir por el lado de aprendér un poco de todo y participar en un equipo más chico con un rol definido pero al mismo tiempo ayudando en

muchísimas funciones diferentes. Serías una especie de comodín y podrías responder a varios problemas de diferente índole.

Volviendo a como nos vemos desde afuera, también quiero mencionar que (inexplicablemente) aún hoy en día, no está instalada la idea de que es fácil unirse a este clan de la tecnología como desarrollador. Aún hoy en muchos casos nos siguen viendo como algo extraterrestre. ¡Creeme que no es así! Es cada vez más fácil. De hecho, si decidís entre una carrera de grado como Abogacía o Contador Público o Administración de Empresas (por nombrar algunas de las más comunes en Argentina), pienso que vas a tardar mucho más en encontrar un lugar para desarrollarte en el que seas feliz y puedas hacer cosas interesantes a que si lo hicieras en sistemas.

¿Por qué pienso de esta forma?

En el mundo IT y casi en ningún otro, las carreras universitarias y terciarias buscan actualizarse constantemente con mejores currículas de enseñanza, constantemente aumenta la accesibilidad de cursos en línea, las herramientas vienen cada vez más preparadas para ayudarnos en el proceso de programar, cada vez hay foros más reconocidos y confiables (o los que hay son cada vez mejores) y sumado a todo esto, estamos en una etapa de recambio generacional en donde vemos cada vez más mentores y mentoras dispuestas a ayudarte en cada paso. Todo este combo quiere decir que en menos tiempo (y con menos inversión económica) vas a poder empezar a obtener resultados. ¡Y qué resultados! La satisfacción de tener un software o app en la que fuiste de alguna forma parte activa en su desarrollo, y que ese producto tenga el potencial de llegar a cientos -por qué no soñar, y miles o millones- de personas, da que pensar.

Dato de color: ¿Sabías que podés cambiar la vida de las personas con una app? Más adelante voy a contarte algunos ejemplos de apps orientadas a mejorar la calidad de vida de personas discapacitadas.

Para mí es muy claro: Si mezclás el tiempo y las cosas interesantes que podés hacer, el mismo disminuye si tu preparación es de sistemas y mucho más si estás acompañado y mentoreado de forma correcta.

No es que esté en contra de carreras de grado o de profesionales que no sean de sistemas, simplemente pienso que la oportunidad de despegar más rápido es muy notoria. Conozco personas maravillosas que no se dedican a sistemas y no por eso no despegaron. También conozco personas de sistemas a las que les costó tanto o más que a una persona no-sistemas. Es algo relacionado a la carrera, al entorno, y al final y lo más importante, a las personas. Entonces, que quede claro: ¡Vas a tener que esforzarte!

Intentando hacer una comparación sana y los más subjetiva posible, a continuación voy a enumerar algunos títulos sobre los cuales baso mi afirmación y el título de este capítulo: Porque es tan interesante el mundo del desarrollo.

Porque faltan cada vez más recursos

La historia nos ha enseñado que con cada revolución tecnológica (es decir, con apariciones de nuevas tecnologías que impacten de manera profunda en la sociedad), la necesidad de personas especializadas que se necesitan para hacer frente a la demanda aumenta progresivamente. Con cada cambio tecnológico disruptivo, y que de alguna forma afecta a la sociedad, el mercado tecnológico toma de esa misma sociedad gente especializada y empuja a que se generen nuevas formas de lograr habilidades.

Actualmente estamos transitando cambios tecnológicos sin precedentes, con computadoras con capacidad de procesamiento inmensas, inteligencias artificiales creadas dignas de películas de los 90 y máquinas robot con ventajas demasiado obvias como para no tenerlas en cuenta: disminución de accidentes de trabajo, mejoras en procesos, disminución de costos, aumentos de producción y cientos de cosas más por las que deberemos aprendér a convivir con ellas.

La cuestión es muy particular, ya que las nuevas generaciones crecieron prácticamente rodeadas de este tipo de tecnologías como usuarios y consumidores, pero al mismo tiempo aún no hicieron el clic y no se ven generadores de productos y servicios. Al seguir pensando de esta forma, como consumidores, sigue predominando la elección de carreras tradicionales y un poco saturadas en países como Argentina. A todo esto y como forma de respuesta, las casas de estudio hacen por lo general un esfuerzo titánico por atraer recursos y mantenerlos, hay cada vez más cursos online en donde el abanico de aprendizaje es inmenso, y las empresas conscientes de lo grave de la situación buscan por todos lados atraer, desarrollar y fidelizar talento. Además, existen diferentes iniciativas, tanto en el ámbito público como en el privado, en donde ya existe una conciencia del problema y se buscan diferentes soluciones, pensadas tanto en el corto como en el largo plazo. Por nombrar algunas y ponerte en órbita:

En el ámbito público, una de las últimas iniciativas conocidas fue el plan 111Mil[2], orientado a la formación de 100 mil programadores, 10 mil profesionales de grado en carreras afines y 1.000 emprendedores tecnológicos, con la condición de ser mayor de 18 años y tener el secundario completo. Se buscó implementar a través del dictado de cursos en universidades seleccionadas como sedes a lo largo del país.

Los objetivos del plan son:

- Formar a jóvenes de todo el país como futuros programadores, para dar respuesta a las demandas del sector empresarial y productivo en el campo del software, así como también para ser precursores de sus propios emprendimientos.

- Impulsar la acción conjunta de las universidades, ministerios y empresas para el desarrollo de las capacitaciones.

- Promover la generación y transmisión de conocimientos de rápida aplicación y alto impacto en el desarrollo local y nacional.

Otra iniciativa, que al igual que el Plan 111mil hoy aparece como “Convocatoria Cerrada” en la web oficial, es el Plan Argentina Programa[3].

Como último ejemplo de este tipo de iniciativas a nivel público y ya específico de la Provincia de Córdoba podemos mencionar el Programa Clip[4], el cual se llevó a cabo en 2020 y buscaba crear 1.000 nuevas oportunidades de empleo de calidad y mejorar la competitividad de las empresas tecnológicas de Córdoba.

En el ámbito privado, en 2020, Mercado Libre y Globant, dos unicornios y empresas emblema del rubro presentaron un programa diseñado junto a Digital House, para la inserción laboral en la industria tecnológica de Latinoamérica[5].

La formación tiene una orientación práctica basada en metodologías ágiles para que los estudiantes adquieran las herramientas necesarias para ser parte del mercado laboral tecnológico de la región. La cursada tiene una duración de dos años y los estudiantes podrán especializarse en Front End Specialist o Back End Specialist. Este programa en particular es prometedor, ya que los y las estudiantes no sólo van a aprendér cuestiones técnicas para especializarse, sino que van a tener la posibilidad de recibir acompañamiento con personas del ámbito.

---

[2] https://www.argentina.gob.ar/educacion/secretaria-politicas-universitarias/programa-111mil

[3] https://www.argentina.gob.ar/preinscribirse-en-el-plan-argentina-programa

El curso se realizará de forma virtual a través de una plataforma interactiva desde el 2 de noviembre hasta el 20 de diciembre. Tiene una carga horaria total de 60 horas, por lo que vas a tener que dedicarle como mínimo 9 horas por semana. Los encuentros serán asincrónicos, de manera que puedas manejar tus tiempos teniendo en cuenta tus posibilidades. La capacitación es gratuita y no es necesario tener conocimientos previos.

Al finalizar la capacitación, vas a poder rendir un examen el 21 de diciembre y obtener un certificado que valide tus conocimientos, avalado por el Ministerio de Desarrollo Productivo de la Nación y la Cámara de la Industria Argentina del Software (CESSI).

Contenido

1. Fundamentos.
2. Programación imperativa.
3. Programación con Objetos.

[4] https://www.cba.gov.ar/programaclip/

Objetivo Específico

o Incorporar 600 cordobeses en nuevos puestos de trabajo para insertarse laboralmente en el sector de software.

o Incrementar las habilidades y competencias de 400 trabajadores de empresas tecnológicas cordobesas, generando espacios de formación profesional.

o Contribuir al desarrollo profesional y la movilidad del capital humano entrenado para que continúe trayectos educativos formales

[5] https://www.certifiedtechdeveloper.com/

Todas estas son iniciativas importantes en la búsqueda por mejorar este déficit de recursos, pero corremos el riesgo de que no sea suficiente, y la situación es alarmante si nos ponemos a pensar que el ritmo de digitalización y de necesidad de recursos capacitados y especializados es elevado. El problema de raíz es, a mi entender, que en muchos países nos demoramos en enseñar cuestiones técnicas, de lógica y programación en etapas tempranas del desarrollo cognitivo de la persona, dígase, jardín y primario.

¿Por qué no adelantarnos y cambiar ese paradigma?

En ese caso, no podemos seguir exigiéndole a nuestras maestras y maestros, *profes* y demás involucrados en la educación de los y las más chicas. Tenemos que trabajar juntos, hacer alianzas duraderas entre Estado, empresas y escuelas, y trabajar mancomunadamente para lograr resultados en el largo plazo.

Porque está cambiando la forma de enseñar (y de aprendér)

La educación que recibí yo en mi adolescencia, orientada a copiar y pegar conocimiento en la mayoría de las materias, está un poco obsoleta. Y no es culpa de alguien en particular, creo. Profesoras, profesores y la mayoría de las y los docentes que conozco se preocupa realmente por sus estudiantes. Les preocupa que les vaya bien, que lleven una vida sana y que estudien para los exámenes. Y ahí está el problema. Que estudien para los exámenes.

Los exámenes en general están preparados para repetir algo que vimos brevemente, algo que no tiene una dosis de creatividad, ingenio e innovación. Son métodos de enseñanza en serie y que deberían actualizarse a métodos de crecimiento personales y por equipos en ambientes preparados para que cada uno logre desplegar su potencial y trabajar sobre algo que sienta es un diferencial y su pasión. En cambio, seguimos insistiendo con un modelo que no busca que cada persona se diferencie sino que premiemos al que recuerda más conceptos o se equivoca menos en una ecuación. Acorde a estos cambios, el rol del docente debería mutar, debería acompañar y debería preparar a cualquier estudiante que lo necesite. Pero para esto necesitamos cambiar la forma en que enseñamos la docencia, caso contrario estamos pidiendo a personas que asuman un compromiso inmenso, más del que les cae sobre sus espaldas hoy.

La etapa media educativa, o primaria, ya debería ser un buen lugar para enseñar de forma obligatoria y a quienes decidan seguir ese camino (o a quienes mamá, papá o quien esté a cargo decida en realidad), materias relacionadas a las TICs. En mi caso optamos por enseñarle a nuestro hijo más grande algunas plataformas de desarrollo de código preparadas para su edad, y darle la posibilidad de experimentar y decidir juntos si es algo que le gustaría ver un poco más a fondo. Es un primer pantallazo que necesitan tener, al igual que matemáticas, ciencias naturales y varias más.

¿Te imaginás preguntándole al pequeño de 8 años de la casa cómo crear una fórmula en Excel porque no sabés por donde arrancar? ¿O pedirle a tu sobrina de 9 que te ayude a imprimir un informe porque para lo único que usaste un USB es para cargar el celular? Bueno, eso en mi opinión es el nuevo analfabetismo digital. Y es un problema que arrastramos desde hace mucho tiempo: preparamos personas que se reciben de secundarios sin saber usar la negrita de Word, o sin haber abierto nunca un email. Este tipo de habilidades se dan por sentadas en pleno año 2021. Ya prácticamente se descartan curriculums en donde las únicas habilidades técnicas que figuran son “Office principiante” u “Excel avanzado”. ¿Avanzado? No lo creo, mostrame una macro que calcule algo y ahí podré determinarlo, caso contrario, sabes usar la fórmula “SUMAR”, algo que un gran porcentaje de la población en búsqueda laboral activa hace. Eso no es un diferencial.

Y acá tenemos otro problema: El diferencial. Si seguimos buscando que las clases se abarroten de pensamientos lineales y encasillados, si seguimos buscando corregir exámenes de memoria y no cambiamos la forma de invitar a pensar soluciones, a presentar proyectos y a compartir ideas, vamos a seguir vertiendo cemento en un molde cuadrado que después, lleno de sesgos y preconceptos, a las personas se les hace muy difícil cambiar.

Un modelo clave que deberíamos seguir en las casas de estudio y de forma obligatoria para empresas del rubro TIC es el de pasantías mensuales y con evaluaciones finales, de esta manera lograríamos una sinergia valiosa y una colaboración real.

La colaboración y sinergia entre empresas del sector, casas de estudio, Estado y estudiantes es para mí un punto clave para terminar de cambiar la forma de enseñar.

Porque existe un nuevo paradigma: Mentores y líderes en lugar de jefes. Escuelas en lugar de empresas.

Fue una transición lenta pero de mucho valor la que se fue dando en los últimos años, pasando de tener equipos descentralizados y poco ágiles a equipos comprometidos y trabajando de la mano entre diferentes áreas, ya no de forma aislada, para lograr productos mínimos y salir al mercado. Hablemos de esa transición: hace mucho tiempo, los equipos de IT no eran diferentes de otros en cuanto a que existía la figura de Jefa, Líder, Gerente y muchos títulos para puestos de mandos, en donde lo principal era mantener la imagen, respetar la burocracia y ser lo menos flexibles y transparentes que se pueda. Total, al equipo le doy trabajo y si no les gusta que se vayan. Hoy ese paradigma y esa idea de anti-unidad cambió y los ambientes de trabajo son increíblemente mejores. Los proyectos se logran en mucho menos tiempo, con personas más comprometidas y logrando un equilibrio vida trabajo asombroso.

El trabajo de una persona que lidera un equipo, -y si alguna vez te toca estar en ese lugar acordate de esto-, es crear un ambiente en donde cada uno pueda desarrollar sus habilidades. Ese entorno va a estar liderado, pero él o la líder no van a ser los únicos que hagan que funcione. Esa persona va a ir trabajando como hormiga dejando que las demás personas participen y logren objetivos juntos. Esa persona va a ser la responsable de la sinergia que va a lograr que te desarrolles o que busques urgente otro lugar para hacerlo.

Cuando empieces, vas a necesitar un o una mentora. No es obligatorio y vas a poder desarrollarte igual si no lo tenés, pero miralo como tener un apoyo de alguien que ya vivió muchas de las cosas que estás por vivir, y que puede abrirte puertas en un futuro. Encontrar a alguien que te mentoree no es fácil, el proceso puede durar muchos años, más si tu círculo de contactos relacionados a la materia es limitado. Vas a tener que persistir y no rendirte hasta que encuentres una o varias personas que además de líderes quieran ser tus mentores. Buscá a personas que estén dispuestas a invertir su tiempo (y a veces su dinero) en vos y tus resultados, que crean en vos como persona y confíen en tus habilidades. Una vez que la encuentres, te darás cuenta de que esa persona (si elegiste bien) dista mucho de las tareas que cumplís en tu rutina laboral diaria, probablemente sea una persona con pensamiento y responsabilidades un poco más estratégicas y no disponga de tiempo ilimitado para vos y el seguimiento de tus proyectos. En ese caso, no olvides que esa persona no lo hace con mala intención, sino que carece del tiempo. Es una cuestión de lógica y que va a servirte para que aprendas a forjar una relación en donde no se vea saturada por vos, sino que vea que la llamás cuando realmente la necesitás. Generalmente esa persona no va a responder tus preguntas y hacerte todo tan fácil, sino que va a enseñarte formas de resolver los problemas, va a guiarte y tratar de trabajar con tus habilidades potenciándolas y generando nuevas. No le preguntes cosas que podrías fácilmente encontrar en internet. No le hagas preguntas técnicas que podrías hacerle a algún colega. Tratá de que te ayude con indicaciones en la forma de encarar el camino, y no esperes que tenga todas las respuestas, pero sí esperá sinceridad y confianza de su parte.

A propósito de este cambio de paradigma, cambió la forma en que las empresas cuidaban a su gente. Ya no existen las cavernas, así que podés sacarte esa imagen de prehistoria en donde desarrolladores son azotados con látigos en un ambiente lúgubre y oscuro para llegar a la última fecha de entrega (tampoco tan así pero viene bien la dramatización para que te lo imagines). Todo lo contrario. Hoy las empresas proveen espacios verdes, iluminación a medida, salas de recreación con pool, metegol, videojuegos, salas de descanso, plantas en cada rincón y muchísimas cosas que hacen que te sientas a gusto y quieras darlo todo en la cancha -virtual- para seguir sumando en esa sinergia. Y de repente no es más una empresa a donde vas a ir a trabajar, cumplir horario y volver cansado y fastidioso con tu jefa o líder, sino que vas a ir a una escuela, una escuela nueva, donde vas a aprendér todos los días, a equivocarte, a tener recreos copados con tus compañeros, en los tiempos que vos y los demás decidan y a dejar de estudiar de memoria para ese examen, sino a probar cualquier *feature* nueva con una forma distinta a la anterior. Eso es el mundo de la programación.

Por la rápida salida laboral

Sí, y cada vez más y más rápida. Entre programadores contemporáneos a menudo nos preguntamos cómo hacía la gente hace 30 años para desarrollar sistemas, sin internet, foros especializados o tener 5 compañeros al lado. Yo me formé en la etapa transitoria, en donde rara vez necesitábamos buscar en libros (aunque a los que nos gusta leer, hayamos leído algunos por puro placer), pero tampoco teníamos la cantidad inmensa de información que hay hoy en internet. O por lo menos, nadie nos había explicado cómo buscar, y qué tan fácil era. Para situarte en contexto: cuando empecé aún no se hablaba de Web Responsive. Hoy Google lo toma como estándar para cualquier desarrollo *web-based*. Recién en ese momento Bootstrap empezaba a surgir como algo que iba a pisar fuerte. Por lo tanto, creo que era un poco más difícil ingresar al mercado laboral.

Hoy, en donde las principales plataformas de educación online rondan en los miles de cursos orientados a programación, es una etapa maravillosa en donde podés empezar a trabajar o tomar algunas horas de un proyecto después de ver algunos cursos y teniendo al día la carrera. ¡Es increíble!

La buena noticia es también para las empresas: esta creciente globalización y predisposición de contenidos de acceso fácil, rápido y gratuito, hizo que puedan armar equipos de trabajo que respondan a proyectos de muchísimas tecnologías diferentes.

Pensemos ahora en cualquier otra carrera. ¿Es posible que esa persona que conozcas, digámosle amiga, amigo, prima o hermano, tenga la posibilidad de probar sus conocimientos con tan poca preparación, con algunos cursos vistos en internet como complemento de su carrera, digámosle, dos años cursados? Probablemente no, y eso es uno de los grandes diferenciales de esta carrera, y por qué no, producto de la falta de recursos, la falta de regulación (¿escuchaste a muchos desarrolladores contarte que necesitaron sacar su matrícula para tipear código?) y muchos otros factores más.

En este punto, nobleza obliga, tengo que hacer una aclaración: no dejes tu carrera en una universidad o terciario pensando que los cursos en línea son la solución. En mi opinión, no lo son. Son un complemento muy valioso, pero no lo son todo. La base está en el aula, haciendo *networking* con profes, escuchando charlas de invitados, teniendo la posibilidad de discutir con compañeros y compañeras que en un futuro podrían ser colegas y mucho más. No pierdas esa oportunidad hermosa de acceder a una carrera pensando que tu futuro está asegurado con cursos *online*.

Porque es un mercado en crecimiento

A diferencia de otras profesiones y de otros mundos en donde puedas desarrollarte, el de IT está en constante crecimiento, es dinámico, no se detiene. Este dato no es menor si aún no estás seguro de qué forma vas a poder crecer o que habilidades te gustaría adquirir (¿se pueden todas?) dentro de lo qué es el desarrollo de sistemas. Si comparamos con otras carreras, la de sistemas tiene la particularidad de pertenecer a un entorno que constantemente está mutando y cambiando. El ritmo de salida a la luz de nuevas tecnologías es increíblemente alto y eso produce que con cada nuevo cambio tecnológico, se empiece a generar un mercado de oferta y demanda en donde los únicos capaces de dar soporte a esa demanda son las personas que se especializan en esa tecnología específica nueva.

El uso masivo de la web, la normalización de un celular por persona con varias aplicaciones instaladas, el uso de Smart tvs y pantallas inteligentes en los autos, o el uso de Smart Watch para salir a trotar 3kms y alrededor de tu casa, y muchos cambios más que se dieron en diferentes campos de aplicación, desde sensores en animales hasta sensores en plantillas de personas diabéticas, dan como resultado que el crecimiento del mercado tecnológico sea un incierto y la única constante que nos atrevemos a declarar es que va a seguir en alza, indefinidamente.

Porque existen diferentes formas de trabajar

Si aún no te convenciste, también quiero que tengas en cuenta el hecho de que en un futuro tal vez puedas trabajar desde cualquier lugar del mundo, en el horario que vos elijas, y con la ropa que vos quieras (sí, estoy hablando de pijamas de ositos y crocs), y esto, por suerte, es una situación cada vez más normalizada.

A mí personalmente me hubiese gustado intentar ser un nómade digital. Está en el ideal de muchas personas el viajar y trabajar, y poder vivir por un tiempo viajando. ¿Ya podés imaginarlo, no? Vos, tu perro orejudo, tu *notebook* y una van preparada en el medio de la montaña con una taza de café al lado.

Ok, volviendo a la realidad, la verdad es que hoy, a raíz del COVID-19, muchas empresas decidieron darle remoto indefinido y a elección de la persona. ¡Es un *game changer*!

En su momento, era difícil hacer un *matcheo* entre freelancers y empresas que necesitaban y requerían que el trabajo fuese de forma dependiente y presencial, por lo que muchos recursos quedaban afuera de grandes empresas y otros lo hacían a través de subsidiarias como prestadores de servicios. Este cambio claramente llegó para quedarse. Llegó para mejorar la calidad de vida de muchísimas personas relacionadas a sistemas, y adivina qué pasa cuando las personas relacionadas a sistemas están contentas. Los productos y servicios que generan son en consecuencia mejores desde muchísimos aspectos.

Mi consejo en este punto es que si buscás ser un/a freelance autónomo o trabajar remoto para alguna empresa, no dejes pasar las oportunidades que se te presenten. Puede ser para una empresa cumpliendo un horario de forma remota, o para varias y cumpliendo hitos. Si ves que la prueba no da resultado, entonces podés probar haciendo presencial. Como dije, las posibilidades en desarrollo son infinitas.

ENCAJE PERSONA EMPRESA

A menudo, cuando vemos películas o leemos historias, nuestra cabeza empieza a divagar con caminos alternativos y a imaginarnos finales diferentes en donde por ejemplo, Harry Potter elige Slytherin o el Capitán América pelea defendiendo a Hydra. En ese caso, ¿hubiese conseguido el buen mago preadolescente seguir por un camino de bondad y defendiendo a sus amistades, a la vez de hacerse un especialista en su campo? ¿Steve Rogers hubiese tenido las posibilidades de crecimiento que le dio su bagaje con el equipo de Vengadores?

Situaciones así en las que podríamos divagar con escenarios alternativos hay miles y la cuestión de fondo -para bajarlo un poco a la realidad- es: ¿va a darte la empresa o la escuela que elijas las herramientas necesarias para ser el superhéroe o la heroína en la historia profesional de tu vida?

Si ya decidiste incursionar en el mundo del desarrollo, o ya sos parte de este, tal vez te hayas preguntado algo así como:

¿Y si no me gusta la empresa?

¿Realmente hay tanta demanda como para animarme a dejar un puesto de trabajo estable?

¿Tendré la posibilidad de hacer base en esa empresa?

¿Qué se necesita para desarrollar alguna habilidad específica?

¿Son todas las empresas tan permisivas?

¿Existe un balance entre lo que la empresa quiere que haga y lo que puede ofrecerme como retribución?

Para todas estas preguntas válidas, para mí hay una sola respuesta y la resumí en unas siglas a las que les puse EPE (Encaje Persona Empresa).

Tal vez hayas leído o escuchado acerca del Encaje Producto Mercado, el cual se define -básicamente- como la forma en que un producto encaja en un mercado, es decir, es cuando una *Startup* ha sido capaz de descubrir un problema relevante para un grupo considerable de clientes y para los cuales han comenzado a prototipar una solución[6]. Ese descubrimiento puede deberse a tener las cosas claras de antemano, conociendo el mercado objetivo y haciendo énfasis en conseguir más adoptantes tempranos que validen esa solución, o, directamente salir a buscar adoptantes y validar. En el ámbito de *startups* [7]es un punto necesario para armar un modelo de negocios prolijo, pero trasladándolo un poco a nuestro proyecto, acá no estamos validando si nuestra solución sirve sino buscando la forma de determinar si tenemos un buen encaje en la empresa que estamos actualmente o en la que queremos o podemos ser parte. De la misma manera que salimos a buscar adoptantes tempranos en un *startup*, mi sugerencia es conocer de la forma más educada posible a personas que trabajen para la empresa o para el equipo. Por lo general, los valores de las personas son fáciles de determinar una vez iniciada una charla. Algunos tips que podrían servirte como *insights* en esas charlas:

- ¿La pasión brota por todos los poros o es desgaste e inconformidad acumulados lo que se manifiesta?
- ¿La radio pasillo sólo transmite rumores o la música es calma y se pueden escuchar personas intercambiando ideas y conexiones?
- ¿La empresa y los líderes de cada área reciben a sus nuevos empleados y empleadas contándoles su historia, sus sueños, compartiendo objetivos y metas, poniendo a disposición tecnologías y herramientas, o al contrario, la empresa siente que es obligación de la persona adaptarse? No te olvides que atrás de un nuevo miembro de equipo hay una persona, con miedos, inseguridades y temores. Siempre es un punto a favor si la empresa demuestra interés en ayudarte en el proceso de aterrizaje.
- ¿Los y las integrantes de la empresa -o del equipo en cuestión- tienen valores morales bien establecidos y respetan a diferentes géneros, identidades y perspectivas?

[6] https://en.wikipedia.org/wiki/Product/market_fit

[7] https://es.wikipedia.org/wiki/Empresa_emergente

- ¿Existe una llegada sana a tus superiores y superioras y de ellos podés aprendér o es una utopía en esa empresa?

- ¿Existen actividades extralaborales como reuniones, partidos de futbol, vóley, juntadas a comer, y cosas por el estilo?

- ¿Tiene la empresa un Plan de Carrera o Desarrollo adecuado a medida de lo que necesitan sus colaboradores para crecer y la educación es continua o es algo que se va viendo sobre la marcha y generalmente no existe la posibilidad?

- ¿Las reglas de juego son claras y se respetan para ambos lados, tanto empleado como empleador?

Dejando un poco de lado lo que es cultura, vamos a lo específico de tecnologías. Si prestás atención, el EPE puede ser aplicado a empresas de cualquier rubro y no sólo de base tecnológica, por lo menos hasta este punto:

- ¿La empresa realiza revisiones técnicas, code reviews o algo similar para corregir constantemente lo realizado y mejorar las bases de los proyectos? Si es al contrario, cada proyecto surge, se implementa y no se aprendé de los errores de este, entonces es un problema, ya que nunca podrán establecerse de manera prolija mejoras.

- ¿La empresa usa o está en sus planes utilizar nuevas tecnologías? Si bien muchas compañías tienen sistemas desarrollados con tecnologías un poco más viejas, está bueno ver si está en sus planes el reemplazo de estos por nuevos lenguajes y arquitecturas, o al menos está en el plano el encarar proyectos nuevos de esta forma.

- ¿Las decisiones técnicas son tomadas por personas idóneas? Si al contrario son tomadas por una persona con el cargo de Líder de Proyectos, o similar -los títulos pueden ser varios- que poco sabe de cuestiones técnicas y de tiempos, estimaciones, vulnerabilidades y desarrollo en general, más un sinfín de cuestiones técnicas que sólo alguien con un poco de experiencia en el rubro puede tener, es probable que en el mediano y largo plazo sea un problema. Otra situación común es cuando se da la falta de liderazgo formal. En estos casos, el liderazgo está compartido y las personas terminan teniendo puntos de vista totalmente diferentes. Al no existir la figura de mediador o guía que resuelva esas diferencias, es más probable que el caos sea moneda corriente.

- ¿El equipo técnico lleva una cultura de aprendizaje en conjunto, documenta lo que es necesario y hace lo mejor posible para que en el futuro sea menos costoso volver a tocar ese código?

- ¿La empresa provee herramientas de trabajo acordes al nivel de resultados que buscan? Computadoras, sillas ergonómicas, escritorios, oficinas limpias e iluminadas y más. Parece un error estar mencionando este punto, pero la verdad es que aún es normal que en muchos lugares te vayan a pasar cosas como que te pidan que lleves tu *notebook* para ir a trabajar.

Todo esto es difícil de determinar el día cero, el uno y probablemente el 60. Pero a medida que pase el tiempo son respuestas que van a ir apareciendo y vas a ir notando cómo van cambiando tus ideas preconcebidas. Hacer todo este análisis de forma regular, digamos una vez al año, va a permitirte establecer algunos puntos que vas a querer tener en claro respecto al EPE de tu persona y de una empresa en particular. En algunas ocasiones puede sucederte que no obtengas un buen resultado de EPE, pero que a la vez no tengas la posibilidad de hacer ese cambio, por cuestiones económicas, de localización o de crecimiento. En esos casos te sugiero que sigas leyendo y trabajes en conseguir un buen estado de salud mental y desarrollo profesional. Otras veces podría suceder que si bien el resultado EPE no es bueno, no sea culpa de la empresa (ni tuya), podría suceder que aún no conociste a fondo a tus compañeros, los valores de la empresa o a los líderes de esta. En ese caso también es válido saber que hay que darle un tiempo prudente de aprendizaje al proceso de conocer variables y personalidades que a veces parecen ocultas o están sesgadas por algún tipo de razonamiento. A veces no es bueno pensar

constantemente en cambiar, porque esto podría derivar en una actitud mucho más tendiente a no tolerar y a estar en alerta en lugar de relajado. En la próxima sección vamos a ver un poco en desglose a dónde puede llevarte la tendencia a mantener esa actitud.

¿Qué diferencia a los Juniors de otros Senioritys?

En el caso que optes por desarrollarte en alguna empresa, en el siguiente cuadro voy a mostrarte algunas convenciones genéricas que se esperan en Juniors. De esta manera podrás evaluar y comparar con tu realidad. No pierdas de vista que esto es genérico y en cada equipo podrías encontrarte con leves diferencias, inclusive podría pasarte que en muchas empresas no tan desarrolladas no cuenten con un plano de carrera armado, por lo que va a depender de vos y de cuánto esfuerzo quieras dedicarle a esa empresa sin saber específicamente los diferentes puntos donde podés ir haciendo base y mejorando tu seniority.

	Junior	Semi Senior	Senior
Experiencia laboral	Menor a 18 meses	18 m. a 5 años	Mayor a 5 años
Conocimientos técnicos	Nulo/Bajo	Autosuficiente	Referente del área
Conocimientos funcionales	Nulo/Bajo	Maneja circuitos lo suficientemente para trabajar de forma autónoma	Ayuda a definir procesos, metodologías y estándares.
Proactividad	Necesita que lo ayuden constantemente	Pide asignaciones nuevas cuando tiene tiempo libre y lo aprovecha.	Busca y genera nuevos requerimientos, es quien genera nuevas posibilidades.
Seguimiento requerido	Requiere seguimiento casi a cada hora	Requiere seguimiento semanal	Reporta sólo las tareas en las que está trabajando
Productividad, Calidad y Eficiencia	Nula/Baja	Media/Alta	Alta
Cumplimiento de fechas	Pocas veces cumple	Casi siempre cumple con fallas mínimas	Siempre cumple
Respuesta bajo presión	Se bloquea, angustia, confunde, estresa.	Entiende los problemas e intenta resolverlos con lo que conoce, no pierde el foco	Resuelve los problemas

STRESS/BURN OUT

Hasta ahora vimos todas ventajas y características sumamente interesantes del mundo del desarrollo y me imagino que ya estás buscando cursos en línea o contactándote con esa profe de programación que sabías que laburaba en una grosa empresa para que te tire unos buenos tips, y porque no, te haga un lugarcito en su equipo cuando pueda.

Bueno, las olas a veces arrastran cosas que no es agua celeste, cristalina y en donde se refleja el sol de la tarde. Tenemos un lado super delicado en esta profesión: el psicológico.

En 2020 la OMS clasificó al Burn Out o Síndrome del Quemado como un trastorno mental y sugiere que debemos tratarlo como otras enfermedades[8].

Imaginemos la siguiente situación: un equipo de 10 personas está desarrollando la nueva *feature* que va a permitir que todas las personas usen el *home banking* con huellas digitales. Sí, son cajeros que se desbloquean con huella digital como el *celu* y no con una clave. Adiós para siempre el acordarte de esos cuatro numeritos.

Ahora bien. La salida a producción, como se dice en la terminología para indicar que esa nueva funcionalidad va a poder utilizarla todo el mundo (o una parte, ya que podríamos ir haciéndolo segmentado), está prevista para el 26 de abril de 2022. Hace 14 días que el equipo viene trabajando sin descanso, haciendo muchas horas extras, tipeando código y haciendo *code reviews* hasta entrada la noche.

Resulta que el 18 de abril nuestro Jefe de Proyecto recibe un *email* indicando que el Banco Central cambió disposiciones a último momento, y además de usar huellas dactilares, vamos a necesitar un segundo factor obligatorio de autenticación para acceder a nuestra cuenta mediante una combinación precisa del juego del Sapito (si sos de mi época tal vez recuerdes a Julian Weich haciéndolo en la tele por las noches, sino, seguro Google te da un gran pantallazo).

Antes de entrar en pánico, entremos en pánico, piensa inmediatamente el Jefe de Proyecto. A pocos días del lanzamiento, y sin posibilidad de cambiar la fecha ni de negociar para una próxima etapa esa funcionalidad, la bajada de línea es clara: Tenemos pocos días para terminar algo impensado. No sabemos que librería vamos a usar para reconocer gestos, no sabemos si tenemos una secuencia o patrón base confiable, no tenemos idea si va a fallar y ser un agujero en la seguridad de nuestra solución y la lista de no sabemos se agranda con cada respiro en la reunión de equipo de la mañana siguiente.

Ahora, imaginate que sos una de las personas involucrada en el desarrollo de cualquier parte de ese sistema. Intentá ponerte en sus zapatos y recreá la imagen en tu cabeza. Una funcionalidad nueva, a nada de terminar la etapa, con un requerimiento que no entendemos del todo, que no vamos a poder probar y que seguramente defina nuestro futuro laboral en ese equipo. ¿Presión? Un poco. ¿Pánico? Bastante.

Lo que sigue es llanto, desesperación, angustia, y cabezas quemadas. Sí, "*Burn Out*".

El ejemplo anterior es dramático, carece de sentido y respaldos técnicos desde varios puntos de vista, y en la realidad las situaciones no son tan extremas como esta. Suele haber una zona gris y los requerimientos generalmente son negociables. Los y las buenas líderes, además, tratan de abstraer al equipo de esas confrontaciones y dejar que terminen con etapas previamente planificadas, salvo caso urgente en donde tengamos que arreglar algo que está roto y se necesita para que siga funcionando el sistema. De todas maneras, es un claro ejemplo de que en esta profesión, por más que nos guste, amemos y deseemos cada día aprendér algo nuevo, llega un momento de quiebre en donde nuestra cabeza decide que, como no paramos antes, le corresponde a ella.

Los síntomas del *burn out* son variados e incluyen, entre otros:

- Sentimiento de agotamiento, fracaso e impotencia, lo que lleva a una inminente baja autoestima.
- Poca realización personal.
- Estado permanente de nerviosismo y dificultad para concentrarse.
- Comportamientos agresivos, impaciencia e irritabilidad.

[8] https://www.who.int/occupational_health/publications/pwh3sp.pdf?ua=1

- Dolor de cabeza.
- Taquicardia.
- Insomnio.
- Bajo rendimiento y absentismo laboral.
- Aburrimiento.
- Comunicación deficiente.

Cómo prevenirlo

Este es uno de los puntos en donde creo que, al ser junior y por lo general tener poco roce con la realidad laboral, más que algunos consejos y un buen acompañamiento de tus líderes, lo mejor es estar siempre atento a las señales del cuerpo y aprendér con la experiencia.

Mis recomendaciones son:

- No te excedas en el ritmo de trabajo y aprendé a utilizar y respetar las pausas para los descansos y la comida.
- Aprendé a establecer límites ante situaciones en las que haya un exceso de tareas, un escaso margen de tiempo para llevarlas a cabo o deficiencias en los medios al alcance.
- Cultivá un buen ambiente de equipo entre tus colegas, promoviendo la colaboración y una mejora en el clima laboral.
- Cuidar de vos mismo es imprescindible, lo que incluye aprendér a atender e interpretar las señales del cuerpo y las emociones que aparecen. Así, por ejemplo, ante la señal de tensión puede ser beneficioso que te permitas un momento de descanso para relajar y respirar profundamente antes de retomar la tarea que estabas realizando.
- Separá el ámbito laboral del personal, aprendiendo a desconectar al finalizar la jornada laboral realizando actividades que lo faciliten.
- Buscá apoyo familiar y social.
- No dudes en buscar la ayuda profesional de un psicólogo.

En el caso de que seas parte de un equipo y el mismo se desempeñe en un ambiente sano, en constante crecimiento, con colegas que trabajan en sintonía y se respetan ciertas reglas de convivencia además de laborales, podríamos decir que va todo viento en popa. Si, por ejemplo alguno de los ítems que tomaste para hacer tu conclusión de EPE no está dando bien al cabo de un tiempo en la empresa, o si constantemente se agregan nuevas *features* a esa etapa que estás por largar y no se tienen en cuenta tiempos, ni descansos, ni nada por el estilo, probablemente necesites reevaluar un par de puntos e intentar no llegar hasta el punto de quiebre.

Mi Experiencia

En mi caso, me sucedió dos veces en donde aún luchaba por hacer un equilibro entre mi vida personal y laboral (y tengo que admitir que es una lucha y un aprendizaje constante), en donde la balanza se inclinaba claramente por la laboral. En esas situaciones de stress, exceso de trabajo y de planificación deficiente, y me vi auto forzado a cumplir con entregas a las que ya me había comprometido (el término auto forzado hace referencia a que, en ambas oportunidades, podría haber negociado fechas de entrega posteriores, pero decidí no hacerlo, pensando que de alguna forma podía con todo), mi cabeza dijo basta y mi cuerpo me lo notificó.

En esas dos situaciones, muy similares y en el mismo trabajo, mi nivel de rechazo hacia las personas subió considerablemente, mis respuestas eran cortas y llenas de ira, y mi humor era pésimo. Un compañero de oficina y de equipo para el olvido digamos. Además de eso, sentí taquicardia y sensaciones

de angustia permanente y durante varios días. Es una situación horrible para personas jóvenes que amamos lo que hacemos, y con la información de toda esta sección lo que más quiero es que no te pase lo mismo.

Tratá de salir, desenfocarte, tener actividades afuera de la oficina, ser paciente y no querer comerte el mundo de un bocado la primera semana. Es una profesión hermosa para no cuidarte y dejar de aprovecharla.

ENTREVISTAS

Dime cómo te entrevistan y te diré qué empresa es.

El proceso de entrevistar personas para un puesto (y digo personas por qué eso es, al fin y al cabo, lo que deberíamos evaluar cuando consideramos puestos de Jrs), es sumamente delicado. Es necesaria mucha práctica para visionar a una persona que tiene pocas medallas y apostar por su futuro. A través de los años me tocó entrevistar algunos Juniors como para lograr sacar algunas conclusiones que espero que te sirvan:

- Presentate prolijo: La prolijidad habla mucho del orden o desorden de una persona. Sé muy bien que no todos somos iguales ni nos gusta usar la misma ropa, peinado o *bijouterie*, pero algo que debemos respetar en cualquier entrevista es una prolijidad mínima. Olvidate del estereotipo del programador desalineado que mira cuatro pantallas al mismo tiempo y desconecta una bomba escuchando heavy metal a todo volumen y en ojotas. La realidad es un poco diferente, las oficinas son lugares de intercambio y donde debe primar el respeto mutuo, así que arrancá por estar respetable, sin perder tu identidad y estilo.

- Respetá los horarios: Del otro lado hay personas ocupadas y revisando muchas opciones junto a la tuya. No restes puntos de antemano dando a entender que va a ser un problema el cumplimiento de algunas reglas como el horario. Algo que podría suceder (en esta profesión y en muchas otras), es que si bien vos respetes los horarios, la otra parte -la entrevistadora- no lo haga. Tené paciencia (y bienvenido/a al mundo real).

- Investigá a la empresa y si podés, a la persona que te va a entrevistar: Saber a qué se dedica la empresa, aunque sea de forma superficial y no tan profunda, pero saber a dónde apunta, cuál es su fuerte, que divisiones tiene, cuántas personas conforman los equipos de trabajo, que tecnologías utilizan y demás, va a ayudarte a estar más distendido en las entrevistas y tener un panorama más claro al momento de establecer comunicaciones. Si además investigás a la persona que te va a entrevistar, tenés la posibilidad de intentar empatizar y llegarle de buena forma, sin cometer equivocaciones hablando de algo que no deberías.

- Escuchá activamente: No es que no tengas que hablar. Es que si estás en el camino de arrancar como junior, vas a tener que escuchar mucho más de lo que hables. No seas una estatua, pero tampoco te vayas de mambo, lo ideal es que muestres pasión y ganas de hacer, de emprender y de animarte.

- No critiques tecnologías ni procesos internos: Me pasó en una oportunidad en que la persona que estaba entrevistando empezó a criticar de forma totalmente aleatoria algunas de las tecnologías que (sin saberlo) en mi equipo usábamos en aquel momento. Me dio la impresión de que no sabía que yo no pertenecía a Recursos Humanos y era el Líder Técnico, es decir, mi comprensión técnica era alta y entendía perfectamente de lo que me hablaba, por lo tanto descarté a esa persona automáticamente y sin hacerle pruebas técnicas porque considero que alguien sin experiencia no puede criticar. Es como querer comparar cosas que no conocés por alguna que otra opinión que escuchaste o que te dijeron.

- Pedí una devolución o respuesta con fecha determinada: Siempre con respeto, no está mal que pidas una devolución. Muchas empresas tienen un seguimiento malísimo en cuanto a devoluciones luego de hacer evaluaciones. Sólo se comunican con vos si fuiste elegido para el puesto, caso contrario ni se toman la molestia. Para no caer en un estado de angustia y nervios por días sin saber si van a hablarte o no, podés pedir qué te digan entre que fechas esperar una respuesta, sea negativa o positiva.

- No dejes de ser vos mismo/misma: Jamás. Si la empresa no está de acuerdo con tu forma de ser, entonces no es la empresa para vos.

Perfiles en Redes Sociales y Repositorios

Linkedin, Stackoverflow y repositorios como GITHUB y GITLAB, por mencionar algunos de los más conocidos, son las formas más comunes de darse a conocer siendo Junior y teniendo poca experiencia y una red de contactos limitada. Por lo tanto, va a depender de vos y de que tanto actualices tus conocimientos (y posteriormente tus perfiles en línea) para empezar a formar una gran telaraña que pueda servirte en un futuro. Hacé cursos en línea, intentá desarrollar proyectos propios o unirte a otros. De esta forma vas a estar dándote a conocer y la persona que esté del otro lado va a ver que no sólo te animás, sino que podés codear siguiendo reglas y estándares, aun siendo Junior.

Podés engancharte en proyectos de otros equipos y hacer ramificaciones propias (véase Forks[9]) o solicitando que unan cambios a la rama principal, en donde resolviste algunas *issues* particulares (véase Pull Request[10]).

Linkedin es muy particular, ya que es la red social profesional con más crecimiento y aunque no es puramente IT, es lo que predomina. Por eso la incluí y creo que no deberías perder tiempo y empezar a armar tu perfil, agregar contactos y actualizarla constantemente. Como cualquier red social en un mundo hiper globalizado y donde las fronteras virtuales no existen, nunca sabes qué tipo de relación puede asegurarte. De paso, podés empezar a trabajar en ser parte activa de la comunidad desarrolladora, una actividad con una sinergia increíble y que siempre suma en momentos en que necesitás una mano para resolver algo o en qué estás buscando compañeros de equipo para proyectos, o cuando querés aprendér algo nuevo.

[9] Un Fork o Ramificación es una copia de un repositorio. Bifurcar un repositorio te permite experimentar libremente con cambios sin afectar el proyecto original. https://docs.github.com/es/github/getting-started-with-github/fork-a-repo

[10] Fusionar una solicitud de extracción dentro de una rama ascendente cuando el trabajo está completo. Cualquier persona con acceso de escritura al repositorio puede completar la fusión. https://docs.github.com/es/github/collaborating-with-issues-and-pull-requests/merging-a-pull-request

TECNOLOGÍA COMO IMPULSOR DEL CAMBIO

La tecnología es un recurso. Un apoyo. Como tal, los que nos encargamos de trabajarla y desarrollar productos y servicios en base a ella, tenemos que, de alguna forma, encontrar un punto de equilibrio en donde sintamos que estamos haciendo cosas que valgan la pena y mejorando el mundo basándonos y apoyándonos en ella, y no solamente malgastando horas tipeando código. También vamos a estar haciendo negocios y viviendo de esto, por supuesto.

La tecnología tiene el poder de transformar vidas enteras, desde personas hasta sociedades. Por ejemplo, en los últimos años, vimos cómo el desarrollo dejó de ser algo exclusivo del género masculino. No es que no existieran otros géneros o no los hubiera en desarrollo, sino que no se fomentaba su inclusión lo suficiente, no éramos conscientes de la gran diferencia de géneros presente (¿o lo éramos y no hacíamos nada para cambiarlo?). En los últimos años se fueron armando diferentes comunidades virtuales a las que podrías acceder para solicitar ayuda. Si bien no estoy relacionado con ninguna, reconozco su valor y puede que me esté olvidando de varias, en ese caso, viene bien una búsqueda:

- Women Who Code https://www.womenwhocode.com/
- Chicas Programando https://twitter.com/chicasprogar?lang=es
- Las de Sistemas https://twitter.com/lasdesistemas
- AsaditoJs https://twitter.com/asaditojs
- Mujeres en Tecnología https://mujeresentecnologia.org/
- Chicas en Tecnología https://chicasentecnologia.org/

Es cierto que las empresas del rubro están madurando mucho más rápido de lo que lo hacen las de otras veredas, siendo pioneras en inclusión de géneros, origen o gustos. Lo más importante es la bondad de la persona como tal y que pueda estar a la altura de los desafíos, lo demás es irrelevante. Pero este no es un tema nuevo, no pienses que estoy tocando el tema géneros para subirme a alguna ola (aunque muchas personas y empresas lo hagan). Si, por ejemplo, no conocés la historia de Alan Turing[11], te recomiendo que la googlees, veas material relacionado en Netflix o leas alguna de sus tantas biografías. El tipo fue un precursor en el campo de la Informática, hay una prueba referida a la Inteligencia Artificial que lleva su nombre[12], tuvo la capacidad de descifrar códigos cifrados de máquinas (en particular los de “Enigma”) interceptando y descifrando con éxito las comunicaciones de la Alemania Nazi en la Segunda Guerra Mundial, y contribuyó profundamente a que la guerra en cuestión fuera más corta de lo que debería haber sido. Aun así, fue humillado y castigado por su homosexualidad. En una época en donde era oficialmente ilegal ser homosexual en Inglaterra (y en muchas partes del mundo), y después de haber contribuido con logros de tal calibre para su nación, le obligaron a elegir entre prisión o someterse a una castración química.

Pero sin ir más lejos, aún hoy la violencia verbal y la burla son moneda corriente contra homosexuales, transgéneros, no binarios y más. Y en el fondo creo que es porque no tenemos la capacidad de ver las cosas con claridad, actuamos por impulso y en base a preconceptos. No estoy hablando de cientos de años atrás, en donde la discriminación era moneda corriente, cuando no podías entrar a una cantina o subir a un autobús público si tu piel era oscura. Estoy hablando del presente y ese ciclo sigue repitiéndose como un bucle. Es tu oportunidad de cortarlo.

[11] https://es.wikipedia.org/wiki/Alan_Turing

[12] La prueba de Turing o test de Turing es un examen de la capacidad de una máquina para exhibir un comportamiento inteligente similar al de un ser humano o indistinguible de este. https://es.wikipedia.org/wiki/Prueba_de_Turing

Si hablamos de inclusión y discapacidad, vemos que existen muchas empresas que están cambiando las formas de hacer las cosas, ofreciendo propuestas innovadoras para lograr la inclusión real en personas con distintas discapacidades. Y no estoy hablando de un simple *slogan* de campaña o una moda para que las campañas publicitarias sean mejores, estoy hablando de acciones concretas, provenientes del seno mismo y de la creación de estas empresas, que nacieron y crecieron para responder a problemas específicos. ¿A dónde encontrarlas? Probablemente no sea necesario que viajes mucho para toparte con alguna y presentarte. Desde donde estés leyendo este libro, con una búsqueda en internet vas a ver qué en algunos kilómetros a la redonda podés encontrar alguna. Acá va una recopilación, que no sigue ningún orden y tiene como objetivo que abras tu cabeza y veas en qué se está trabajando y a donde podés aportar:

- OTTAA Project: OTTAA Project es un Sistema Aumentativo Alternativo de Comunicación, destinado a personas con discapacidad en el habla. Es una herramienta móvil, rápida y efectiva que mejora significativamente la calidad de vida y facilita la integración social y laboral. [13]

- Hablalo: Es una app que permite que todos y todas nos comuniquemos, haciendo hincapié en personas con dificultades en su comunicación. Es de las más conocidas y su Fundador, Mateo Salvatto, es un gran referente en el campo de la inclusión mediante tecnología. [14]

- CAECUS: Lentes inteligentes que brindan asistencia remota a personas con discapacidad visual. Dicha asistencia comienza cuando se insertan en entornos urbanos con problemas de accesibilidad. [15]

- TUR4all: Se trata de un buscador que tiene la finalidad de ayudar a la planificación de un viaje. Para ello, el buscador muestra la información de los establecimientos y las actividades sobre la accesibilidad física, auditiva, cognitiva y visual que puedan tener las personas. TUR4all permite, por ejemplo, encontrar hoteles que tengan una buena accesibilidad para las personas en sillas de rueda o transporte adaptado. [16]

- iDentifi: Esta aplicación tiene su público objetivo en las personas invidentes. Enfocando la cámara del teléfono móvil frente al objeto en cuestión, realiza una descripción de audio de dicho objetivo. [17]

- Dilo en señas: Está creada no sólo para las personas con discapacidad auditiva sino también para todas aquellas que quieran aprender el lenguaje de señas. Ofrece el aprendizaje de la lengua de signos recurriendo a un juego.

- Disabled Park: A esta aplicación pueden recurrir todas las personas con movilidad reducida. Permite localizar en las ciudades los aparcamientos reservados para ellos. Disabled park presenta un mapa de la ciudad donde indica las plazas de aparcamiento. [18]

- uSound por Samsung: Aplicación móvil que te permite hacer un test auditivo y conocer tu riesgo de hipoacusia. [19]

---

[13] https://www.ottaaproject.com/

[14] https://hablalo.app/

[15] http://caecuslab.com/

[16] https://www.tur4all.com/

[17] http://getidentifi.com/

[18] https://www.disabledpark.com/

[19] https://www.usound.co/es/

Existen muchas soluciones a nivel nacional, no dejes de investigar este tipo de cosas. Hoy en día, muchas me siguen llamando la atención, por su simpleza de desarrollo por un lado, y por su robustez a la hora de solucionar problemas y llegar a muchísima gente. ¿Ya te dije que la tecnología cambia la vida de las personas, no?

Mi Experiencia

Personalmente y hablando de géneros, cambió mi forma de ver las cosas en los últimos años. Pude ir de a poco cambiando preconceptos que tenía muy arraigados desde la época preadolescente. El clic puede darse en cualquier momento, creo que al intentar ser padres y líderes responsables, necesitamos cambiar nuestra forma de ver las cosas y aceptar ideas que no son nuestras. En mi caso fue una cuestión de aprendizaje e introspección, en el que tuve que dedicarme a hablar con personas bien informadas en otras posturas y leer mucho sobre temas que antes no lo hacía. Entendí que debíamos cambiar la forma en que comunicamos las cosas en sistemas, no sé si sea mucho, pero es un comienzo, al fin y al cabo somos comunicadores y debemos fomentar la unión, el intercambio de ideas y la mejora continua y no el odio y la polarización.

En cuanto a discapacidad, nos tocó de cerca por tener una hermana con TGD (Trastorno General del Desarrollo)[20], pero a diferencia de otras familias, nuestros padres pudieron estar presentes en cada momento, y explicarnos desde pequeños que el camino es la inclusión y no la exclusión. Creo que toda esa mentalidad está reflotando y haciéndose eco al momento de trabajar el tema géneros.

[20] https://es.wikipedia.org/wiki/Trastorno_generalizado_del_desarrollo

Capítulo 2
HABILIDADES BLANDAS

QUÉ ESPERAN DE VOS

En esta sección voy a darte un pantallazo de algo que seguramente te preguntaste en algún momento y que es un tema de gran controversia en muchas carreras, pero especialmente en la del desarrollo.

Si mi experiencia es poca o nula, ¿qué esperan que haga?

Si la mayoría de las veces voy a equivocarme, en mayor o menor medida pero voy a hacerlo, ¿qué hago? ¿Cómo van a evaluarme?

¿De qué depende el seniority del puesto?

La situación, como yo la veo, es algo así: por un lado, tenemos estudiantes recién recibidos o a punto de hacerlo y que con poca experiencia práctica necesitan empezar a desenvolverse y probar lo que pudieron aprender. Por otro lado, tenemos empresas -y no todas- que hoy en día siguen cometiendo errores cuando salen a contratar recursos juniors:

1. Buscando personas con poca o nula experiencia y pretendiendo que lideren grandes áreas y equipos o sean responsables de cambios de raíz. Sí, es más barato a corto plazo, pero están haciendo agujeritos en la base de sus botes y en algún momento va a hundirse.
2. Pidiendo más años de experiencia en algunas tecnologías o herramientas de las que tiene la herramienta misma. Parece ilógico, pero es real. Las áreas de captación y contratación deberían trabajar más en conjunto con áreas tecnológicas.

Al margen de estas situaciones, que no está de más comentarlas ya que podrías vivirlas, quiero contarte que es normal que te sientas abrumado en los primeros días en tu carrera en el mundo IT. La mayoría de las veces vemos herramientas que desconocemos, lenguajes, terminología, y otras cosas que jamás escuchamos y de las que seguro tenemos muchas dudas.

¿Escuchaste algo de Middleware? ¿API? ¿Qué es eso de “commit” y “push” a una rama específica?

Okey... Vamos de a partes y con calma.

En el próximo capitulo (Habilidades Técnicas), veremos con detalle un poco más de terminología. Por ahora quiero comentarte qué buscamos en Juniors los y las líderes de equipos de desarrollo:

- Compromiso y actitud: Pensá en un equipo de desarrollo como un equipo de básquet de alto rendimiento (por mencionar algún deporte). Nadie quiere en su equipo una persona poco comprometida, que llegue tarde a los entrenamientos, que coma mal durante los días previos a los partidos y que no quiera practicar tiros libres, que tan mal le salen. En definitiva, nadie quiere un compañero con poco compromiso, holgazán y al que no le importe el éxito del equipo. En el desarrollo es lo mismo, sólo que no tiramos al aro, sino que clonamos repositorios y resolvemos *issues*. La actitud no es negociable. Muchas veces va a suceder que tengas que cumplir con algunas tareas de más de las que te habían asignado, o tengas que hacer algo para lo que no te contrataron o inclusive tengas que adelantar en investigaciones de nuevas formas de resolver las cosas. Es todo parte del aprendizaje y forma tu carácter. Y creeme que los verdaderos líderes tienen en cuenta este tipo de actitudes, no hace falta que lo menciones en voz alta. Siempre hay alguien que sabe lo que estás haciendo (o lo que no).

- Prolijidad: Si bien es un tema que tocamos en la sección de entrevistas, no está de más mencionarlo. En mis equipos de trabajo invito a que limpien y ordenen sus espacios de trabajo al finalizar el día. Eso te va a dar una claridad al día siguiente y va a permitir que te enfoques en los temas que tenés que resolver. También recomiendo que lleven con orden y prolijidad sus programas de gestión de tareas, etapas y mucho más. No es que sea un fanático del orden, creo que en la mezcla hermosa entre el orden y el desorden (conocido también como entornos

caórdicos[21]) surgen ideas geniales sin malgastar tanto tiempo y esfuerzo, y creo que es una combinación que da resultados siempre que no estés en los límites.

- Comunicación: En muchos casos los y las juniors son aún muy jóvenes y no tuvieron roce laboral, por lo que su comunicación tiende a no ser tan clara y precisa como la de alguien que hace un tiempo se dedica a esto. En este punto creo que lo mejor es adoptar una comunicación en donde predomine la escucha activa, en donde el interlocutor no tenga que repetirte las cosas porque no le prestaste atención y desde la cual vas a poder empezar a mejorar tus habilidades orales y escritas. Intentá ser los más claro posible usando pocas palabras pero particulares a los temas que querés tocar, y no te enredes con palabras técnicas que desconocés sin antes buscarlas, esto pasa seguido y es malo para tu crecimiento porque ya vas a estar forjando una tendencia a repetir sin antes investigar.

- Auto medición y reconocimiento de errores propios: No esperes a que tu líder te diga que estás haciendo algo mal. Buscá formas de evaluarte a menudo e intentá mejorar tu trabajo siempre mirándolo desde una óptica diferente a la tuya. De esta manera vas a estar haciendo una autoevaluación no tan subjetiva. Reconocer tus errores y trabajar en mejorarlos o pedir ayuda es muy importante. Vi muchas veces cómo desarrolladores con potencial hacen las cosas mal, no porque no tengan la capacidad de hacerlo bien, sino porque no tienen la capacidad de pedir ayuda. Consecuentemente, el tiempo utilizado es más alto de lo que debería, y el código no es de calidad.

- Perder el miedo a equivocarse: Escuché de varias empresas que les pagan a sus equipos grandes bonificaciones por proyectos presentados y posteriormente implementados. De esta forma se crea una cultura colectiva de aprendizaje y se anima a la gente, sin importar el rango jerárquico, a que presenten ideas innovadoras. Si bien existen varias compañías de este tipo, puede que en donde estés no sea así, pero el principio de permitir las equivocaciones y que logres perder el miedo debería primar.

- Jugador de equipo: ¿Te considerás una persona que juega en equipo? Si tu respuesta es negativa o tenés dudas, te recomiendo que empieces a buscar ayuda. El camino del dev es largo y con muchos altibajos, como si fuera una competencia constante en donde necesitamos apoyarnos y confiar en nuestros colegas para llegar con fuerzas a cada juego. Una buena forma de empezar a ser mejor jugador de equipo es desarrollando las habilidades que vamos a ver a lo largo del capítulo.

- Humildad: Nadie quiere personas poco humildes en sus equipos de trabajo. Vas a necesitarla para adquirir nuevos conocimientos, partiendo de la base de que todo lo que sabemos es poco, para estar abiertos a nuevos cambios y opiniones y como forma de transitar el camino del desarrollo.

- Empatía: Empatía con otras personas del equipo, sin olvidarte lo más importante: somos personas. Podemos tener días malos, días en que no nos salen las cosas y en que retrasemos un proyecto, días en que tu líder no pueda darte la atención que requieras y mucho más. No olvides que fuera de la burbuja en que vivimos hay un mundo inmenso. Empatía además con usuarios no técnicos, si te toca tratar con ellos y ellas, vas a notar que a veces es una montaña cuesta arriba. No siempre, pero vas a tener que trabajar tu empatía y entender que no todas las personas son nativas digitales. Y si querés sacarles jugo, más vale que aproveches cada momento que puedas.

- Paciencia: Con vos mismo y con otros. Como vimos hasta ahora, en diferentes puntos, vas a equivocarte muchas veces y vas a hacer las cosas mal (por más que no quieras). Las buenas empresas y los buenos equipos van a tener tu inexperiencia en cuenta y van a intentar asignarte tareas menos duras para que entiendas cómo funcionan las cosas. Eso es esencial en las primeras etapas en el mundo del desarrollo: que entiendas cómo funcionan las cosas, y por si no te quedó claro: vas a cometer errores. Muchos. *Merge Request* que no resuelvan nada (y rompan algo), bases de datos borradas, estándares no seguidos al pie de la letra y mucho más. La mayoría de los errores que cometas ya los cometió tu líder de proyecto, así que no te olvides que alguien ya transitó tu camino, de forma más o menos parecida, y tené paciencia. No te preocupes si no

[21] https://en.wikipedia.org/wiki/Chaordic_organization

avanzás lo rápido que quisieras, preocupate si no te movés. Intentá dedicarle horas productivas y no horas a montones, salir del molde, poné al equipo delante de la persona. No pretendas conocer todo al principio.

- Ser más *Lean*: Entre las habilidades importantes que aprendí y que me hubiera gustado aprender más rápido está la de separar lo que no da valor inmediatamente de lo que sí. Este problema no es sólo de juniors, también es algo que se ve en desarrolladores y desarrolladoras que son buenos escribiendo código y de repente tienen un rol más activo en el liderazgo pero siguen pensando operativamente. La metodología LEAN Startup, propuesto por Eric Ries en su libro Lean Startup, nos dice que una de las reglas más importantes es validar conocimiento de forma temprana, ahorrándonos costos y sacando productos de forma más rápida[22]. Siempre doy el mismo consejo a personas que están arrancando. No te olvides que del otro lado del mostrador hay personas que poco les importa si estás trabajando con la última versión de una tecnología o estás siguiendo patrones de desarrollo de *software*. Buscá un equilibrio entre hacer las cosas de la forma más correcta y segura posible, y entregar soluciones constantemente, para eso van a pagarte.

- Ver las ruedas y adaptarlas: Cuando recién comenzaba escuchaba por todos lados, casi como un slogan de campaña: “No reinventen las ruedas”. Hacían referencia a que no debíamos ponernos a desarrollar algo que ya estaba hecho, sino reutilizar constantemente. Para mí, algo más apropiado es: “Descubrí las ruedas y adaptalas”. Para esto existen miles de repositorios abiertos, códigos de ejemplos, *webs* con proyectos y lo mejor, recursos propios de cada lenguaje para lo que necesites solucionar algo, en donde *devs* de todo el planeta suben y actualizan constantemente paquetes para que vos puedas utilizarlos de forma fácil y gratuita. Buscá, por ejemplo, packagist si usás PHP, npm si tu base es javascript y podríamos estar todo el día, pero lo mejor es animarse a *googlear*. Probá con estos ejemplos antes de hacer algo propio, seguro hay una solución que vas a poder tomar y empezar a utilizar y adaptar para cubrir alguna necesidad puntual.

- Especializarse en principios y no herramientas: Lenguajes, herramientas, IDEs y la lista de cosas necesarias para desenvolverte sigue y sigue. Cuando empieces con algo en particular vas a dedicarle un par de años para luego darte cuenta de que la versión que se actualiza cambia por completo la forma en que se hacían las cosas, o tal vez decidiste cambiar de empresa y notás que en otros lados no usan los mismos lenguajes o frameworks y tenés que empezar casi de cero, dejando un poco de lado los años dedicados. Por eso mi recomendación es que no te apegues a herramientas y trates de enfocarte en principios. Vas a ver que los mismos buscan repetirse en la mayoría de los lenguajes que uses y vas a poder transferir conocimiento de una solución a otra de forma sencilla.

- Organizate y Triunfarás: Desechá lo que no sirve o no es tan importante. Priorizá y apoyate en aplicaciones como Trello para gestionar tus tareas diarias y llevar una planificación actualizada y con calendarios si es posible. No sólo vas a poder tener claridad sobre en qué tenés que hacer foco, sino que vas a poder responder más rápido a requerimiento de nuevos desarrollos, sabiendo tu disponibilidad de antemano. A nivel operativo en el día a día, podés usar técnicas como Pomodoro, la cual va a servirte para tener un ritmo de trabajos y descansos seteado por vos y en donde puedas aprovechar tu concentración al máximo. Como ejemplo:

    1. Trabajo 25’
    2. Descanso 5’
    3. Trabajo 35’
    4. Descanso 10’
    5. Vuelvo a empezar.

[22] https://es.wikipedia.org/wiki/Lean_startup http://theleanstartup.com/

ESTIMACIÓN

En mi oficina actual tengo un cartel que dice: “Algunas horas de estimación pueden ahorrarte semanas de programación”. Es un ayuda memoria para recordarme a mí y al equipo lo importante que es estimar, y que va a determinar qué tan eficiente sea el equipo del que formas parte. Tener una buena planificación, de la cual va a ser responsable la persona con el título de mánager de proyectos o similar, es el paso posterior para lograr una buena estimación en las diferentes etapas que componen el desarrollo de un proyecto de software. Si el equipo del cual formas parte respeta estimaciones y las hacen de forma prolija, vas a tener la posibilidad de participar activamente en las estimaciones.

Sea como sea la forma en que lleven los proyectos en la empresa, vas a poder estimar y aprendér a responder a la pregunta: ¿Cómo mido el desarrollo de ‘x’ requerimiento?

Voy a tratar de hacer énfasis en los aspectos que deberías tener en cuenta siendo junior y empezando a estimar. Voy a obviar algunas cuestionas que están más relacionadas al *managment* de recursos y equipos, puesto que no deberías preocuparte por eso (por ahora).

Cómo estimar siendo Junior

Como venimos viendo, es difícil comenzar a estimar sin experiencia. Más allá del método que utilicen en la empresa o la metodología que usen para la gestión de proyectos, voy a darte algunos tips para que puedas empezar a visualizar como calcular esfuerzos.

Para una correcta estimación, lo primero que necesitamos es contar con información histórica. Al principio no vas a tenerla y vas a tener que guiarte con la ayuda que te den otros *devs* o los líderes del área. Super importante en este momento ser prolijo y anotar el esfuerzo real que le dedicaste a cada tarea. Así vas a ir armando tu propia base de información sobre la cual vas a volver y usarla cuando sea necesario.

1. Dividí las tareas que tenés asignadas en componentes más chicos y fáciles de medir. Por ejemplo, si tenés que desarrollar un software web, podrías empezar dividiendo por componentes: Back End, Front End (siendo muy genéricos). Luego volvé a dividir, asignando cada tarea a alguno de esos componentes:

   CRUD Clientes -> Back End

   CRUD Clientes -> Front End

2. Luego, para cada una de esas tareas, volvé a dividirlas en subtareas que creas que tengas que realizar para completarlas:

   CRUD Clientes -> Back End -> Atributos + Métodos (crear, modificar, eliminar, obtener)

   CRUD Clientes -> Front End -> Tabla + Campos + Comunicación con el back

3. Dedicá un tiempo prudente a estudiar si existen recursos y subcomponentes que puedan depender de otros que aún no descubriste. Tené en cuenta cosas como seguridad, escalabilidad, estándares definidos por el área y requerimientos que no están del todo claro. Volvé al punto 1 para cada uno que encuentres y no dejes de consultar.

4. Asigná horas a cada tarea que fuiste encontrando y dividiendo, armá una tabla prolija que puedas modificar de manera sencilla y compartila si es necesario con un líder o senior *dev*. ¡Siempre digital!

5. Tené en cuenta tiempos para escribir test unitarios, probar y reprobar el código. Nadie quiere unir algo con errores a producción, así que mientras más recaudos tomes, y los dejes asentados, mejor es.

6. Al finalizar todo el trabajo de estimación de la etapa o proyecto que te encomendaron, pedile a algún senior o a tu líder que revean las estimaciones juntos, anotando sus *feedbacks* y tratando de comprender en qué podrías haber mejorado.

Inclusive después de seguir estos breves consejos, y de intentar estimar de la manera más prolija y realista posible, va a pasarte que te equivoques, ya sea sobreestimando o subestimando. En cualquiera de los casos, es algo normal en desarrollo y un margen de equivocación deberían permitirte, caso contrario estarían buscando un Senior (que también se equivocaría en sus estimaciones aunque un poco menos).

La siguiente tabla es un ejemplo de algo muy básico que va a servirte para empezar a completar tu base histórica de esfuerzo (en este caso horas pero podrían ser otras variables).

Proyecto	Tarea	Lenguaje	Esfuerzo Estimado (hs)	Esfuerzo Real (hs)	Observaciones
A	1	JS	3	5	
A	2	JS	4	3	
A	3	JS	1	2	
A	4	JS	2	1	
B	1	Python	1	3	
B	2	Python	4	5	
B	3	Python	4	3	

Por qué es tan importante aprender a estimar

- Transparencia: Teniendo información real y visible para todos los miembros del equipo, vas a notar cómo fluye un ambiente de transparencia y de información compartida en donde todos salen ganando y se generan nuevas relaciones proactivas. Ya nadie oculta qué hace, cuánto tiempo tarda en hacerlo o cómo lo hace, sino que todos buscan fines en común.

- Predictibilidad: En cuanto a nuevos requerimientos o proyectos, vas a notar que el contar con datos va a hacer que sea más fácil para vos y tu líder predecir esfuerzos estimados. Así no se trate de tecnologías que hayas utilizado anteriormente o funcionalidades que conozcas a la perfección, ya vas a poder ir armando una base de auto comparación con experiencias previas.

- Aumento de calidad en el producto final: Al trabajar de manera más ordenada, prolija y ya con esfuerzos estimados, podemos hacer foco en lo que realmente consideramos importante *versus* lo que consideramos urgente. De esta forma llegamos a los fines de etapa o entregables con menos *stress*, sintiéndonos seguros de que pudimos dedicarle el tiempo necesario para desarrollar, probar y volver a probar los nuevos requerimientos.

- Aumento de credibilidad del equipo: Frente a clientes u otros equipos de la empresa, a medida que las estimaciones sean más prolijas, vas a estar ayudando a generar un sentimiento de credibilidad y responsabilidad para el equipo del cual formás parte, punto clave para recibir posteriormente apoyo de niveles jerárquicos superiores.

Causas más frecuentes de errores de estimación

Como vimos, vas a cometer errores constantemente a la hora de estimar, aun siguiendo los consejos que puedan darte además de los que leíste en la sección anterior. Te presento una recopilación de algunas de las causas por las que podrías cometer estas equivocaciones, el objetivo es que tengas una visión aún más clara para poder prevenirlas.

- Poca información sobre estándares de desarrollo: Todo equipo de desarrollo debe seguir estándares que se respeten por sus miembros en cada proyecto. La falta de estándares (o el no comprenderlos) lleva a que cada dev tenga ideas diferentes sobre cómo encarar problemas similares, lo que se traduce en más tiempo de revisión, dificultad en seguir el código y en

hacerlo escalable. Siguiendo estándares vamos a tener la misma base sobre la cual poder estimar diferentes requerimientos.

- Requerimientos que cambian constantemente: Una vez definidos los requerimientos, mi consejo es que los trabajes tal como se definieron y acordaron con clientes o usuarios. De esta forma vas a evitar hacer cambios que no estaban previstos y que puedan “inflar” tu número total de esfuerzo real. En algunas ocasiones vemos cómo van surgiendo requerimientos nuevos después de desarrollar alguna funcionalidad puntual y en ese caso lo recomendable es dejarla anotada en la lista de tareas pendientes y volver a estimarla en la próxima etapa.

- Escasa información sobre el proyecto: Si bien en muchas ocasiones sucede que el nivel de incertidumbre es muy alto al principio, hay diferentes técnicas para hacer que vaya disminuyendo en cada entrega. Lo mejor cuando estás arrancando es tener charlas concretas con usuarios o clientes y tratar de sacarte todas las dudas, dejando poco lugar a esas lagunas que van a hacer que te retrases constantemente.

- Trabajar bajo presión: Cuando tenemos una fecha límite de entrega próxima, a veces olvidamos la importancia de estimar y nos metemos a desarrollar sin más. Este es un error de concepto que deriva en exceso de horas utilizadas para desarrollo y pruebas, lo que termina retrasando entregas. Si bien no vas a tener la experiencia de poder saber con exactitud si vas a llegar o no a la fecha propuesta, está bueno empezar a hacer el ejercicio y ofrecer márgenes de entrega entre dos fechas.

HÁBITOS

Las personas aprendémos por imitación, desde muy chicos copiamos lo que está bien y (por desgracia) lo que no está tan bien. Eso explica por qué es tan importante darles a los y las más chicas un entorno sano en donde puedan tomar valores y costumbres y empezar a hacerlos propios. Esto explica también por qué el rol de madres, padres, docentes y tutores es tan importante. Si la tendencia es que las horas se consuman usando aparatos electrónicos y malgastando tiempos muertos, entonces no es raro que los más chicos desarrollen esos hábitos. Si en cambio lo que predomina es la constante búsqueda de desafíos intelectuales, aprendér nuevas habilidades y no estar nunca conforme con lo que sabemos, entonces los resultados serán diferentes.

Sin ir más lejos, y debido a que las personas estamos constantemente imitando y comparándonos con otras, es necesario que desde etapas tempranas de nuestro desarrollo profesional imitemos hábitos saludables a la hora de transitar el camino. Te dejo algunos que considero claves y fáciles de empezar a poner en práctica.

- Aprovechá los descansos que mencionábamos antes para moverte, respirar aire puro, hablarle a algún familiar, amiga, novia o a tu perro. Plantas también cuentan.

- Tené siempre agua cerca. Mantenerte hidratado ayuda a que el cerebro trabaje mejor y a que tu lucidez no disminuya.

- Descansá la vista constantemente. Revisá tu cronograma de visitas al oculista. Probablemente tengas que ajustar a una visita anual.

- Compartí tus problemas técnicos y funcionales. Si tenés compañeros, compañeras o líderes cerca, no dudes en compartirle el problema. Es la mejor manera de desatascarse. En mi caso me ha dado resultado inclusive contarle mi problema técnico a personas que no son del área IT, por lo que ya sabía que tecnológicamente no podrían ayudarme, pero el expresar el problema en voz alta e intentar comunicarlo de forma clara y sencilla, muchas veces prende esa lamparita casi quemada. ¿Escuchaste acerca de #explainlikeimfive?[23] Intentá seguir esa consigna a la hora de hacer enunciados y explicaciones.

- Leé portales de noticias tecnológicas e intentar estar al día con tecnologías y actualizaciones: eso va a hacer que se expandan tus conocimientos y tus ganas de hacer cosas nuevas, diferentes y arriesgadas.

- Compartí: Durante mucho tiempo pensé que las ideas no debían compartirse. Después escuche de un emprendedor: “el que se considera emprendedor debe aprender a compartír sus ideas, de nada sirve decir, un tiempo después y señalando con el dedo, esa persona hizo plata con mi idea”. En el momento en que empezamos a contar nuestras ideas con personas interesadas es cuando logramos esa conexión que tanto necesitamos con roles diferentes.

- ¿Tenés hobbies abandonados? Volvé a dedicarle tiempo y armá una rutina que no puedas mover por cuestiones laborales. Cualquier cosa sirve para que nuestra cabeza deje de pensar en el trabajo: armar trenes, dedicarte a cuidar el jardín, jugar con tus hijos, sacar a pasear a tus mascotas, montar barriletes, armar muebles, trabajar madera, las posibilidades son infinitas.

- Hacé actividad física: es clave. Ayuda a liberar tensiones acumuladas y nos hace estar más sanos y felices. Natación, gimnasio, correr, salir a caminar, yoga. La lista, otra vez y al no ser que algo te lo impida, es interminable. Lo aconsejable por profesionales de la salud son 3 días a la semana, de 1,5 horas cada día de manera intensa.

- Comé sano: En esta y en cualquier profesión, este es un hábito que todos deberíamos seguir desde muy chicos, sin ser extremistas pero sabiendo que comer mal con regularidad va a causarnos problemas de salud y productividad.

[23] ELI o Explicamelo como si tuviera 5, es intentar explicar de manera sencilla algo complejo.

- Visitá periódicamente al médico: ¡Un momento! ¿Este no era un libro que iba a ayudarme a aterrizar en el mundo del desarrollo? ¿Qué hacemos hablando de visitas al médico? Es un consejo 100% válido teniendo en cuenta que queremos lograr una vida sana para rendir posteriormente tanto en las horas que estemos desarrollando como en las horas posteriores en donde necesitemos estar con familias, amigos o solos.

- No abandones tus proyectos tecnológicos personales: Muchas personas tenemos una gran tendencia a empezar cosas y no terminarlas. En el caso de proyectos tecnológicos, es entendible que si pasás una cantidad de horas trabajando para tu equipo, empresa o proyecto para el que te contrataron, no quieras tocar una línea más de código hasta el próximo día. Y en ese caso, vas a seguir dedicándole tiempo a ese proyecto ajeno. Mi consejo es que no dejes de dedicarle tiempo a ese proyecto tuyo. Vas a aprender cosas nuevas y emocionantes y quién te dice, en una de esas te permite en un futuro manejar tus tiempos. En mi caso me entretiene desarrollar apps para mis hijos, haciéndolos participar del proceso de ideación y prototipos en papel, para después ver que desarrollar un sistema no es algo de otro mundo, está a su alcance. También jugamos con Arduino, empezando a tener una noción básica de circuitos, comunicaciones y demás, otro de los puntos que a desarrolladores de código puro como yo nos faltó aprender.

- Encontrá tu momento creativo: Ese momento del día en donde tu cabeza pueda volar y no existan límites. ¡Intentá que sea descansado!

- Gamificá: ¡Hacelo un juego! Buscá la innovación de forma divertida, utilizá recursos que lo hagan más ameno y que te enganchen más.

Capítulo 3
HABILIDADES TÉCNICAS

HERRAMIENTAS

El hecho de que existan infinitas herramientas es una de las consecuencias del crecimiento del mercado, porque a medida que el entorno crece, es necesario (y surgen como producto de esto) nuevas herramientas y tecnologías para dar soporte al desarrollo en todas sus etapas y procesos: *code reviews, deployments*, seguimiento de tareas, preparación de entornos, captura de requerimientos, validación de interfaces, comportamientos y muchos más. Además tengamos en cuenta que a diferencia de otros rubros, en el tecnológico las herramientas se hacen en la misma cocina. Es decir que vamos a tener gente especializada que, al necesitar nuevas herramientas, las genera. También tenemos el uso de licencias a bajo costo, disminuyendo más aún por cantidades adquiridas. Todo esto, sumado a que los equipos de desarrollo usamos (y mucho) herramientas de todo tipo hace que hoy la oferta sea inmensa.

Tanto si trabajás de forma remota como presencial, en equipos chicos, medianos, grandes o individualmente, la oferta de opciones es siempre grande y es difícil recomendar algunas puntuales o hacer un ranking con las mejores para cada problema. Sin embargo, creo que estas son algunas de las que deberías escuchar y con las que podrías cubrir las principales necesidades si te unís a un equipo de desarrollo:

Código

Cuando empecé a desarrollar hace muchos años no conocía muchos editores y usé el que me recomendaron en ese momento: Notepad++. Si bien no es malo para arrancar, hoy no haría esa recomendación e iría directamente por Visual Studio Code. Ese es el que usamos en mi equipo. Se vuelve más poderoso al momento de usar las extensiones correctas y setearlo para los lenguajes que necesites en cada proyecto. Debajo pongo algunas alternativas igual de útiles si sabés manejarlas.

- VS Code: https://code.visualstudio.com/
- ATOM https://atom.io/
- Sublime Text: https://www.sublimetext.com/

Gestores de datos y bases de datos

Un sistema gestor de base de datos es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos.

- Robo 3T: https://robomongo.org/
- DBeaver https://dbeaver.io/
- MySQLWorkbench https://www.mysql.com/products/workbench/

Desarrollo y documentación de API

Tipear código en un editor no es lo único que hacemos al momento de armar una solución. También necesitamos comunicarnos con el *Back End*, o la *API*, a la vez que armamos toda la documentación para que esto sea escalable y cualquier dev involucrado en el *Front* pueda tomar los recursos que necesite. Para esto tenemos varias alternativas y ya empieza a jugar el tipo de licencia que queramos usar, free o freemium (gratis hasta cierto punto, luego se paga). Te recomiendo Postman si trabajás solo, pero también

podés buscar alternativas más baratas y con menos funcionalidades, ya que esta es muy completa para arrancar. Algunas veces inclusive vas a poder armar un Back End de pruebas (si trabajás en el Front End por ejemplo) sin esperar a que desarrolladores y desarrolladoras del back terminen con sus etapas para ir probando.

- Postman https://www.postman.com
- Mocky https://designer.mocky.io/
- Postwoman https://postwoman.io/

Comunicaciones

Estar en comunicación permanente con el equipo es clave si querés mantener un ritmo de productividad alto y no dejar que las cosas se enfríen. Para esto necesitás herramientas que además de ser en tiempo real (obvio), permitan tener un orden con canales, proyectos, usuarios e integraciones con otras herramientas. Las mejores para mí son Slack y Teams, ambas con posibilidad de integrarse a otras apps, y dividír canales como uno quiera, en mi caso prefiero Slack para equipos de desarrollo que no son parte una organización (no tienen licencia de Office) y Teams con su funcionalidad de usar Word, Excel y Power Point ahí mismo, además de integrar videollamadas.

- Slack https://slack.com/intl/es-ar/
- Teams https://www.microsoft.com/es-ar/microsoft-teams/download-app
- Discord https://discord.com/

Gestión de tareas y seguimiento de proyectos

Comunicarnos en equipo es muy importante y también lo es llevar un orden y un seguimiento del proyecto, desglosandolo en tareas, fechas de entrega, personas implicadas y mucho más. Eso es un gran diferencial de esta profesión, en donde constantemente se busca transparencia y claridad a la hora de saber en qué está trabajando la persona que tenés sentada al lado o a miles de kms y en otro horario. Para esto contamos con muchas herramientas pero creo que lo más importante es conocer algunas y tomar lo que te sirva, haciendo las cosas simples y prolijas, ya que cada una de estas vienen con muchas opciones y configuraciones posibles. Empezar con algunos tableros Kan Ban podría ser lo más fácil, en ese caso te recomiendo Trello o Planner.

- ASANA https://asana.com/es/
- Trello https://trello.com/en
- Planner https://www.microsoft.com/en-us/microsoft-365/business/task-management-software

Git

El sistema de control de versiones más utilizado, difundido y requerido. Estas herramientas se basan en el uso de GIT, por lo que te recomiendo que primero aprendas lo básico del mismo, antes de largarte a tocarlas. También podés utilizar terminal de comandos.

- GitHub Desktop https://desktop.github.com/
- Git Kraken https://www.gitkraken.com/
- GitHub Pages https://github.com/
- Git Branching https://learngitbranching.js.org/?locale=es_ES

Recursos, tutoriales y un poco de todo

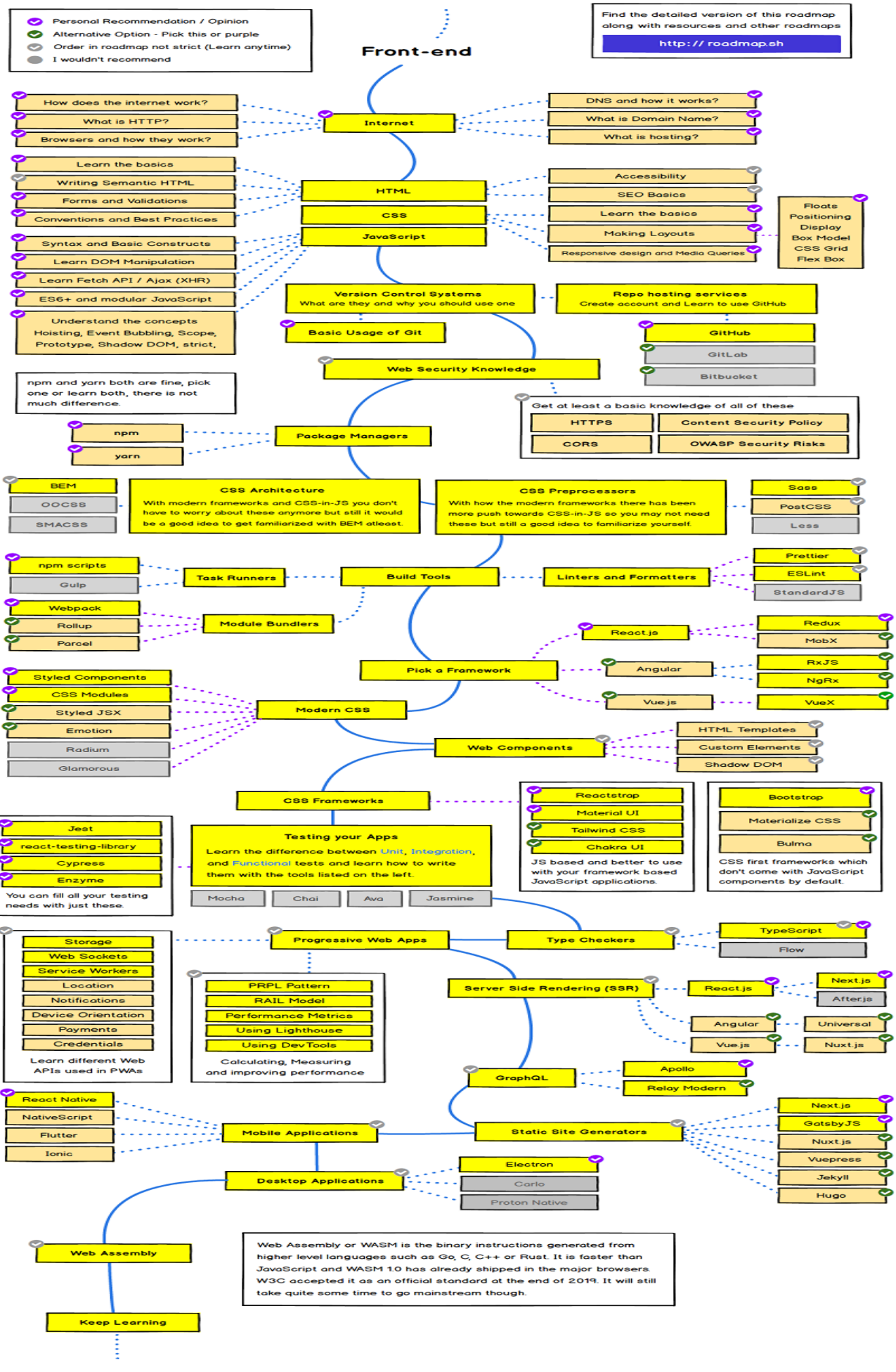
- Netlify: Para construir y hacer deploy de sitios. https://www.netlify.com/
- Amazon Web Services: La mejor nube en cuanto a escalabilidad, prestaciones y seguridad. https://aws.amazon.com/es/
- W3Schools: Amplia variedad de tutoriales web https://www.w3schools.com/
- CodePen: Plataforma para poder compartir y probar Código HTML, CSS y JS en línea. https://codepen.io
- Firebase: SaaS Back End de Google con muchísimas prestaciones (y algunas totalmente gratuitas) https://firebase.google.com/
- Powtoon: No todo es desarrollar. Va a pasarte en algunas oportunidades que necesites mostrar una idea de forma más gráfica y tengas que hacer una presentación rápida. https://www.powtoon.com/

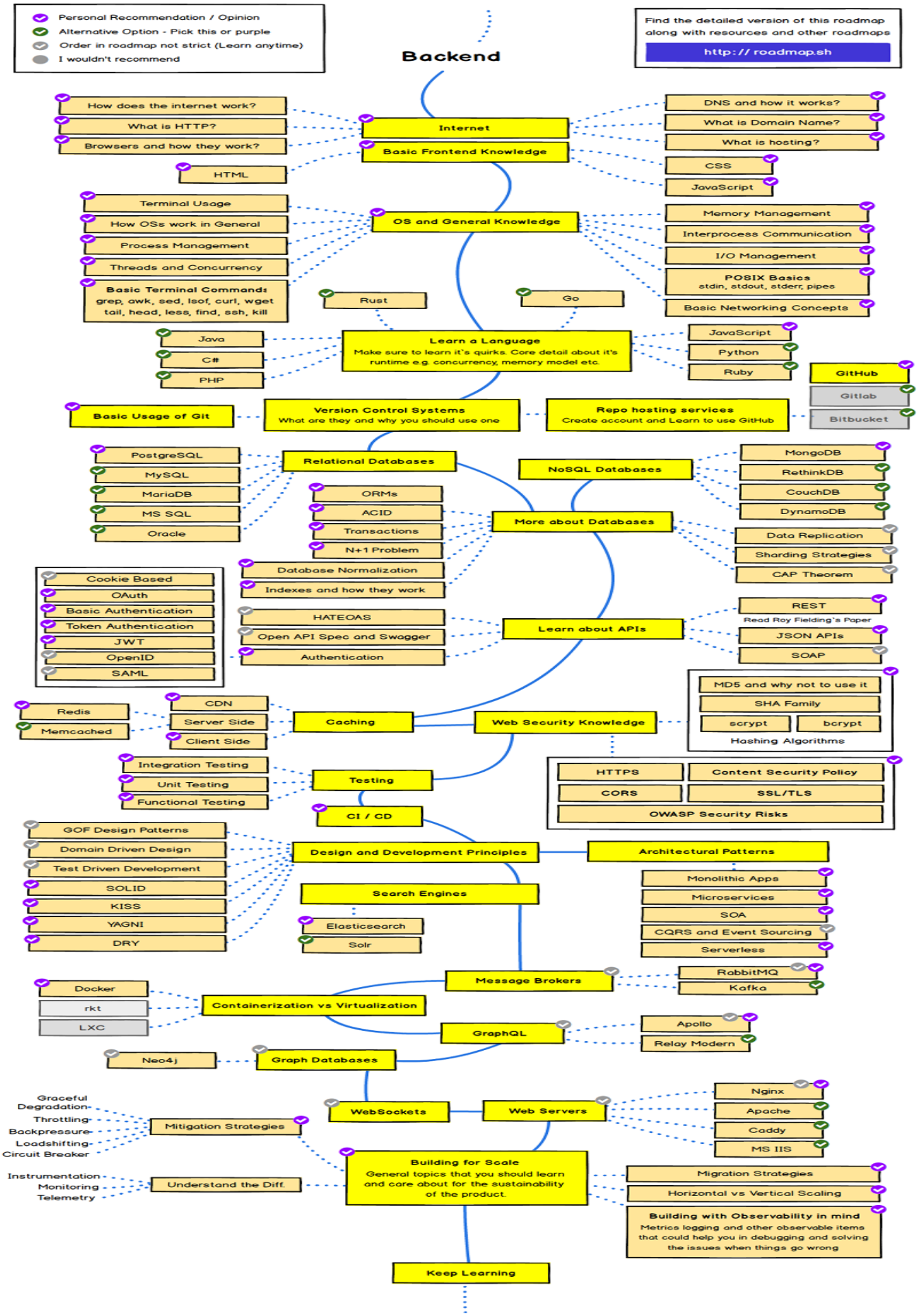
LENGUAJES

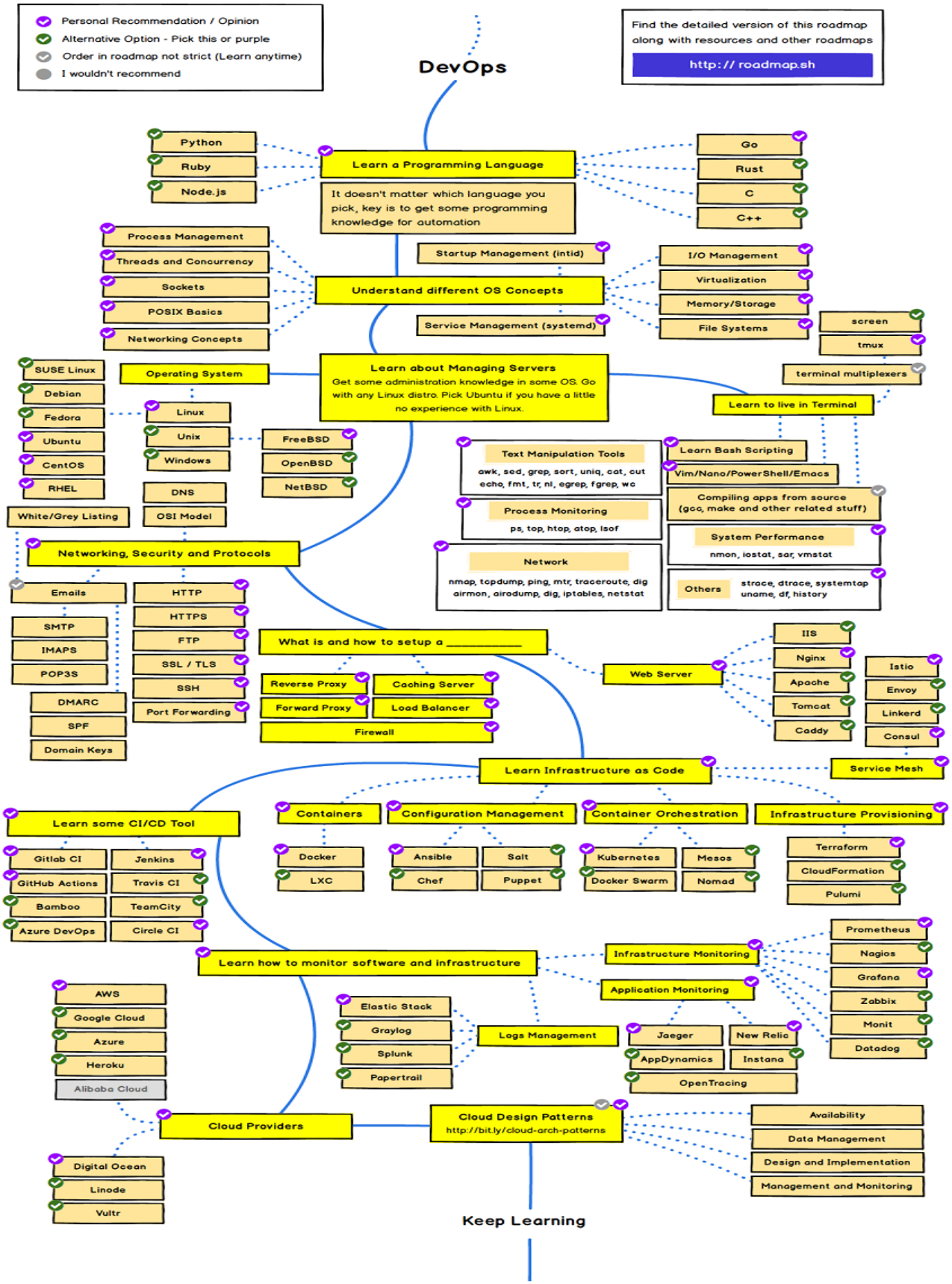
En una época polarizada como la actual, en donde a las personas nos está costando más de lo que debería aceptar otros bandos, otras formas de pensar y de expresarse, el desarrollo no se queda atrás. Vas a encontrarte con devs experimentados (y otros no tanto) que tienen posturas muy definidas y cerradas sobre cuál es el mejor lenguaje para desarrollar en Back, Front o lo que sea y vas a ver que es muy difícil que cambien esa opinión (o al menos escuchen y evalúen otra). Personalmente creo que es un sinsentido y que cada lenguaje, además de estar orientado a alguna función en particular, puede servirnos en diferentes momentos de nuestras carreras, pudiendo dar soluciones distintas cada vez que lo necesitamos. Pensá en las herramientas y lenguajes como una gran caja de herramientas virtual. A medida que conozcas más, aunque sea sólo la superficie, vas a poder resolver diferentes problemas de forma más eficiente. También hay que tener en cuenta la diferencia entre *open source* y de pago, y va a ser algo que seguramente no te toque valorar siendo Junior (a no ser que quieras empezar un proyecto propio), pero está bueno saber que cada elección tiene costos, ventajas y desventajas.

A continuación te dejo tres guías de desarrollo que podrías seguir (están super completas y tal vez te convenga empezar buscando significados de cada tecnología y herramienta en particular, no pienses en aprendér todo junto). Por cuestiones de reutilizar lo que está armado y disponible, considero que el mensaje de tomar algo (con el debido consentimiento del autor) y mostrarlo en un libro es parte del trabajo del programador, en donde vamos a estar usando código ajeno y haciéndolo propio con muchas modificaciones.

Gracias Kamran Ahmed por permitirme incluirlo en el libro. Podés encontrar más del genial trabajo de Kamran en GitHub o en su canal de YouTube


Personal Recommendation / Opinion
Alternative Option - Pick this or purple
Order in roadmap not strict (Learn anytime)
I wouldn't recommend
Find the detailed version of this roadmap along with resources and other roadmaps
http:// roadmap.sh
Front-end
How does the internet work?
What is HTTP?
Browsers and how they work?
Internet
DNS and how it works?
What is Domain Name?
What is hosting?
Learn the basics
Writing Semantic HTML
Forms and Validations
Conventions and Best Practices
HTML
Accessibility
SEO Basics
CSS
Learn the basics
Making Layouts
Responsive design and Media Queries
Floats
Positioning
Display
Box Model
CSS Grid
Flex Box
JavaScript
Syntax and Basic Constructs
Learn DOM Manipulation
Learn Fetch API / Ajax (XHR)
ES6+ and modular JavaScript
Understand the concepts Hoisting, Event Bubbling, Scope, Prototype, Shadow DOM, strict,
Version Control Systems
What are they and why you should use one
Repo hosting services
Create account and Learn to use GitHub
Basic Usage of Git
GitHub
GitLab
Bitbucket
Web Security Knowledge
npm and yarn both are fine, pick one or learn both, there is not much difference.
Get at least a basic knowledge of all of these
HTTPS
Content Security Policy
CORS
OWASP Security Risks
npm
yarn
Package Managers
BEM
OOCSS
SMACSS
CSS Architecture
With modern frameworks and CSS-in-JS you don't have to worry about these anymore but still it would be a good idea to get familiarized with BEM atleast.
CSS Preprocessors
With how the modern frameworks there has been more push towards CSS-in-JS so you may not need these but still a good idea to familiarize yourself.
Sass
PostCSS
Less
npm scripts
Gulp
Task Runners
Build Tools
Linters and Formatters
Prettier
ESLint
StandardJS
Webpack
Rollup
Parcel
Module Bundlers
React.js
Redux
MobX
Pick a Framework
Angular
RxJS
NgRx
Vue.js
VueX
Styled Components
CSS Modules
Styled JSX
Emotion
Radium
Glamorous
Modern CSS
Web Components
HTML Templates
Custom Elements
Shadow DOM
CSS Frameworks
Reactstrap
Material UI
Tailwind CSS
Chakra UI
JS based and better to use with your framework based JavaScript applications.
Bootstrap
Materialize CSS
Bulma
CSS first frameworks which don't come with JavaScript components by default.
Jest
react-testing-library
Cypress
Enzyme
You can fill all your testing needs with just these.
Testing your Apps
Learn the difference between Unit, Integration, and Functional tests and learn how to write them with the tools listed on the left.
Mocha
Chai
Ava
Jasmine
Storage
Web Sockets
Service Workers
Location
Notifications
Device Orientation
Payments
Credentials
Learn different Web APIs used in PWAs
Progressive Web Apps
Type Checkers
TypeScript
Flow
PRPL Pattern
RAIL Model
Performance Metrics
Using Lighthouse
Using DevTools
Calculating, Measuring and improving performance
Server Side Rendering (SSR)
React.js
Next.js
Afterjs
Angular
Universal
Vue.js
Nuxt.js
GraphQL
Apollo
Relay Modern
React Native
NativeScript
Flutter
Ionic
Mobile Applications
Static Site Generators
Next.js
GatsbyJS
Nuxt.js
Vuepress
Jekyll
Hugo
Desktop Applications
Electron
Carlo
Proton Native
Web Assembly
Web Assembly or WASM is the binary instructions generated from higher level languages such as Go, C, C++ or Rust. It is faster than JavaScript and WASM 1.0 has already shipped in the major browsers. W3C accepted it as an official standard at the end of 2019. It will still take quite some time to go mainstream though.
Keep Learning


Personal Recommendation / Opinion
Alternative Option - Pick this or purple
Order in roadmap not strict (Learn anytime)
I wouldn't recommend
Find the detailed version of this roadmap along with resources and other roadmaps
http:// roadmap.sh
Backend
How does the internet work?
What is HTTP?
Browsers and how they work?
Internet
DNS and how it works?
What is Domain Name?
What is hosting?
HTML
Basic Frontend Knowledge
CSS
JavaScript
Terminal Usage
How OSs work in General
Process Management
Threads and Concurrency
Basic Terminal Commands: grep, awk, sed, lsof, curl, wget tail, head, less, find, ssh, kill
OS and General Knowledge
Memory Management
Interprocess Communication
I/O Management
POSIX Basics stdin, stdout, stderr, pipes
Basic Networking Concepts
Rust
Go
Java
C#
PHP
Learn a Language
Make sure to learn it's quirks. Core detail about it's runtime e.g. concurrency, memory model etc.
JavaScript
Python
Ruby
GitHub
Gitlab
Bitbucket
Basic Usage of Git
Version Control Systems
What are they and why you should use one
Repo hosting services
Create account and Learn to use GitHub
PostgreSQL
MySQL
MariaDB
MS SQL
Oracle
Relational Databases
NoSQL Databases
MongoDB
RethinkDB
CouchDB
DynamoDB
ORMs
ACID
Transactions
N+1 Problem
Database Normalization
Indexes and how they work
More about Databases
Data Replication
Sharding Strategies
CAP Theorem
Cookie Based
OAuth
Basic Authentication
Token Authentication
JWT
OpenID
SAML
HATEOAS
Open API Spec and Swagger
Authentication
Learn about APIs
REST
Read Roy Fielding's Paper
JSON APIs
SOAP
Redis
Memcached
CDN
Server Side
Client Side
Caching
Web Security Knowledge
MD5 and why not to use it
SHA Family
scrypt
bcrypt
Hashing Algorithms
Integration Testing
Unit Testing
Functional Testing
Testing
HTTPS
Content Security Policy
CORS
SSL/TLS
OWASP Security Risks
CI / CD
GOF Design Patterns
Domain Driven Design
Test Driven Development
SOLID
KISS
YAGNI
DRY
Design and Development Principles
Architectural Patterns
Search Engines
Elasticsearch
Solr
Monolithic Apps
Microservices
SOA
CQRS and Event Sourcing
Serverless
Message Brokers
RabbitMQ
Kafka
Docker
rkt
LXC
Containerization vs Virtualization
GraphQL
Apollo
Relay Modern
Neo4j
Graph Databases
Graceful Degradation
Throttling
Backpressure
Loadshifting
Circuit Breaker
Mitigation Strategies
WebSockets
Web Servers
Nginx
Apache
Caddy
MS IIS
Instrumentation
Monitoring
Telemetry
Understand the Diff.
Building for Scale
General topics that you should learn and care about for the sustainability of the product.
Migration Strategies
Horizontal vs Vertical Scaling
Building with Observability in mind
Metrics logging and other observable items that could help you in debugging and solving the issues when things go wrong
Keep Learning


Personal Recommendation / Opinion
Alternative Option - Pick this or purple
Order in roadmap not strict (Learn anytime)
I wouldn't recommend
Find the detailed version of this roadmap along with resources and other roadmaps
http:// roadmap.sh
DevOps
Learn a Programming Language
It doesn't matter which language you pick, key is to get some programming knowledge for automation
Python
Ruby
Node.js
Go
Rust
C
C++
Understand different OS Concepts
Process Management
Threads and Concurrency
Sockets
POSIX Basics
Networking Concepts
Startup Management (intid)
Service Management (systemd)
I/O Management
Virtualization
Memory/Storage
File Systems
Learn about Managing Servers
Get some administration knowledge in some OS. Go with any Linux distro. Pick Ubuntu if you have a little no experience with Linux.
Operating System
SUSE Linux
Debian
Fedora
Ubuntu
CentOS
RHEL
Linux
Unix
Windows
FreeBSD
OpenBSD
NetBSD
screen
tmux
terminal multiplexers
Learn to live in Terminal
Text Manipulation Tools
awk, sed, grep, sort, uniq, cat, cut echo, fmt, tr, nl, egrep, fgrep, wc
Learn Bash Scripting
Vim/Nano/PowerShell/Emacs
Compiling apps from source (gcc, make and other related stuff)
Process Monitoring
ps, top, htop, atop, lsof
System Performance
nmon, iostat, sar, vmstat
Network
nmap, tcpdump, ping, mtr, traceroute, dig airmon, airodump, dig, iptables, netstat
Others
strace, dtrace, systemtap uname, df, history
Networking, Security and Protocols
White/Grey Listing
DNS
OSI Model
Emails
SMTP
IMAPS
POP3S
DMARC
SPF
Domain Keys
HTTP
HTTPS
FTP
SSL / TLS
SSH
Port Forwarding
What is and how to setup a _____
Reverse Proxy
Caching Server
Forward Proxy
Load Balancer
Firewall
Web Server
IIS
Nginx
Apache
Tomcat
Caddy
Istio
Envoy
Linkerd
Consul
Service Mesh
Learn Infrastructure as Code
Containers
Configuration Management
Container Orchestration
Infrastructure Provisioning
Docker
LXC
Ansible
Salt
Chef
Puppet
Kubernetes
Mesos
Docker Swarm
Nomad
Terraform
CloudFormation
Pulumi
Learn some CI/CD Tool
Gitlab CI
Jenkins
GitHub Actions
Travis CI
Bamboo
TeamCity
Azure DevOps
Circle CI
Learn how to monitor software and infrastructure
Infrastructure Monitoring
Application Monitoring
Logs Management
Elastic Stack
Graylog
Splunk
Papertrail
Jaeger
New Relic
AppDynamics
Instana
OpenTracing
Prometheus
Nagios
Grafana
Zabbix
Monit
Datadog
AWS
Google Cloud
Azure
Heroku
Alibaba Cloud
Cloud Providers
Digital Ocean
Linode
Vultr
Cloud Design Patterns
http://bit.ly/cloud-arch-patterns
Availability
Data Management
Design and Implementation
Management and Monitoring
Keep Learning

ESTÁNDARES Y BUENAS PRÁCTICAS

Recreemos la siguiente escena: una cocina de restaurant galardonada, con un chef experimentado y de repente estás vos ahí, siendo una persona más en el equipo, ayudando en una tarea específica y demostrando que podés ser una pieza clave.

La primera noche de temporada alta el salón se llena antes de lo previsto y sin reservas, lo que genera un caos en la cocina que el chef trata de apaciguar liderando y mostrando respeto a cada persona, pero exigiéndole efectividad en el armado de los platos.

Tu primer pedido (de la noche y de la temporada), una carne cocinada al horno de barro, sale gomoso. El chef te había explicado (y además está en el libro de introducción a tu puesto) que tenías que cocinarla por un tiempo establecido, manteniendo una temperatura indicada en el horno. Además, la carne debía ser fresca y debía ser adobada previa, durante y posteriormente con hojas frescas de romero y una infusión de hierbas. Como ese día habías descansado poco, estabas con un humor de perros y ánimos bajos, querías hacer las cosas rápido y desligarte de las responsabilidades. Así que decidiste saltearte la mayoría de los estándares del proceso. El (o los) posibles desenlaces de la historia son los siguientes:

1. El chef decide que no estás listo para temporada alta y perdes el trabajo.

2. Perdés un cliente.

3. Aprendés que los estándares están para algo y empezás a usarlos, a respetarlos y a proponer nuevos, mejorando los procesos y creando nuevos.

¿Viste qué fácil es entender y ver las ventajas de usar estándares cuando no hablamos de desarrollo? Acá pasa lo mismo. Es indispensable seguirlos.

Podría haber contado la historia de muchas formas diferentes pero creo que el mensaje hubiese sido el mismo: los estándares sí importan. Gracias a ellos podemos hacer mejor código, más escalable, seguro y trazable, podemos agregar personas al equipo que entiendan qué se hizo hasta el momento, podemos colaborar con otros equipos de desarrollo de forma más fácil y podemos disminuir costos haciendo las cosas más rápido, mejor y sin necesidad de volver a reescribirlas.

Voy a mencionar los más importantes desde mi punto de vista, pero al final del capítulo te dejo los links a Wikipedia (sí, aún creo en el conocimiento libre y espero que los links estén activos al momento de que los necesites), en los mismos vas a encontrar Principios, Patrones y Estándares muy útiles en el desarrollo de software y te aseguro que empezando a aplicarlos de a poquito vas a ser mejor profesional.

KISS

Ya lo había propuesto Leonardo Da Vinci, gran inventor y padre de muchas de las ideas que hoy en día se aplican de diferentes formas: “La simplicidad es la máxima sofisticación”. El principio KISS (Keep It Simple, Stupid!) establece que la mayoría de los sistemas funcionan mejor si se mantienen simples que si se hacen complejos; por ello, la simplicidad debe ser mantenida como un objetivo clave del diseño, y cualquier complejidad accidental debe ser evitada. De esta forma vamos a evitar no sólo código espagueti[24] sino también en errores en diseño que posteriormente repercutan en usuarios que no quieran utilizar nuestros sistemas, poca escalabilidad y mucho más. Además de aplicarlo, es bueno empezar a pensar en forma KISS y tratar de resolver problemas complejos de manera sencilla.

Algunos tips para tratar de cumplir con KISS:

- Mantener métodos y clases pequeñas.

- Usar nombres claros para las variables.

- No reutilizar variables.

- Dividir -siempre- el problema en partes más pequeñas y qué puedas resolver de a poco.

[24] El código espagueti es un término peyorativo para los programas de computación que tienen una estructura de control de flujo compleja e incomprensible. Su nombre deriva del hecho que este tipo de código parece asemejarse a un plato de espaguetis, es decir, un montón de hilos intrincados y anudados.
https://es.wikipedia.org/wiki/C%C3%B3digo_espagueti

- No abusar de comentarios, usar siempre en lugares clave para entender funcionamiento y no para explicar que hace algo específico del lenguaje.

DRY o DIE

Si alguna vez viste código espagueti, seguro que pensaste "¿No deberíamos haberlo resuelto sin repetir estas funciones?" o "¿Por qué tenemos métodos iguales, no podríamos hacer uno solo?". De esto va el principio cuyas siglas significan: Don't repeat yourself, o No Te Repitas. Es una forma de encarar procesos que busca evitar la duplicación de código, argumentando que "cada pieza de información" nunca debería ser duplicada debido a que la duplicación incrementa la dificultad en los cambios y evolución posterior.

Cuando el principio DRY se aplica de forma eficiente, los cambios en cualquier parte del proceso requieren cambios en un único lugar. Por el contrario, si algunas partes del proceso están repetidas por varios sitios, los cambios pueden provocar fallos con mayor facilidad si todos los sitios en los que aparece no se encuentran sincronizados.

También se lo conoce como DIE (Duplication is Evil).

SOLID

Este acrónimo fue introducido por Robert C. Martin[25] a comienzos de la década del 2000 y representa cinco principios básicos de la programación orientada a objetos[26] y al diseño. Cuando estos principios se aplican en conjunto es más probable que el sistema sea fácil de mantener y ampliar con el tiempo, de esta forma podremos eliminar malos diseños provocando que tenga que refactorizarse el código fuente hasta que sea legible y extensible. En mi opinión es el más importante ya que, a diferencia de KISS o DRY, propone cinco reglas que todo dev debería seguir para respetarlo. Los ejemplos que le siguen a cada principio son genéricos y podés encontrar el post en idioma original en https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

Para la captura de código utilicé Carbon https://carbon.now.sh/

Principio de responsabilidad única (Single responsibility principle): Este principio establece que una clase sólo debe tener una responsabilidad sobre el total de la funcionalidad, es decir que para contener el cambio debemos tener claramente definidas las responsabilidades. Otra forma de entenderlo es que una clase sólo debe tener un motivo para cambiar, lo que significa que sólo debe tener una tarea.

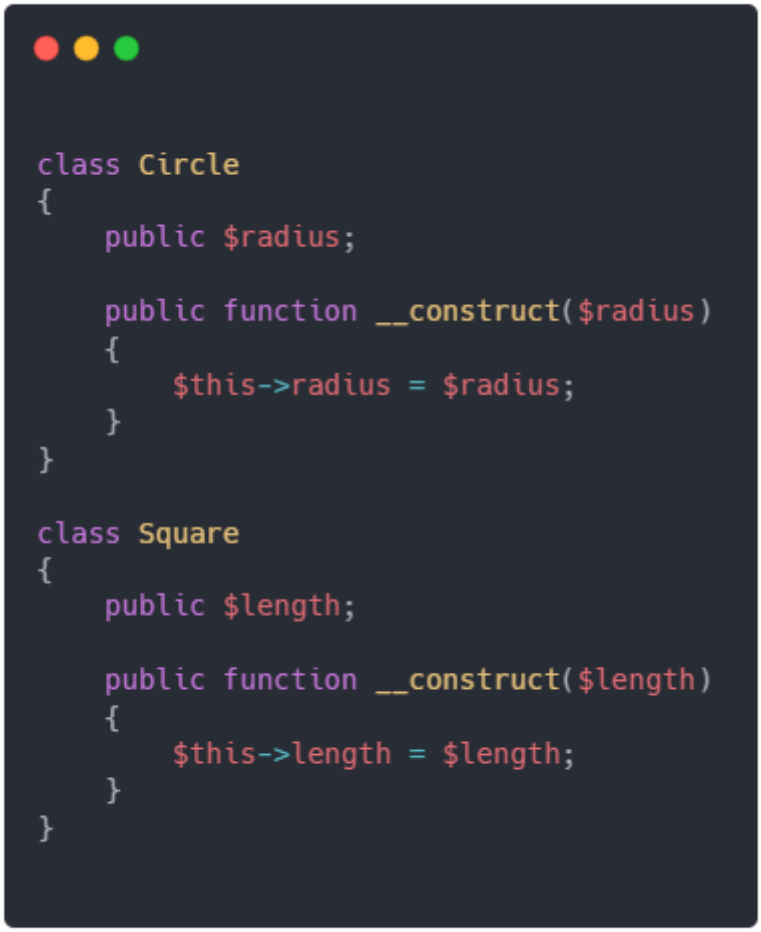
Ejemplo:

Tenemos varias figuras de las que después queremos calcular su área total. Primero creamos las clases de las figuras y dejamos que los constructores se encarguen de recibir las medidas necesarias.

[25] Robert Cecil Martin (n. 1952, coloquialmente conocido como Uncle Bob) es un ingeniero de software y autor estadounidense. Es coautor del Manifiesto Ágil. Ahora dirige una empresa de consultoría llamada Uncle Bob Consulting LLC y Clean Coders, que aloja videos basados en sus experiencias y libros. https://es.wikipedia.org/wiki/Robert_C._Martin

[26] La Programación Orientada a Objetos (POO, en español; OOP, según sus siglas en inglés) es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos se utilizan como metáfora para emular las entidades reales del negocio a modelar.

Está basada en las siguientes técnicas: herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento. https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos



```php
class Circle
{
    public $radius;

    public function __construct($radius)
    {
        $this->radius = $radius;
    }
}

class Square
{
    public $length;

    public function __construct($length)
    {
        $this->length = $length;
    }
}
```

Ahora creamos la clase AreaCalculator, que recibe un *array* con los objetos de cada una de las figuras para ser sumadas.

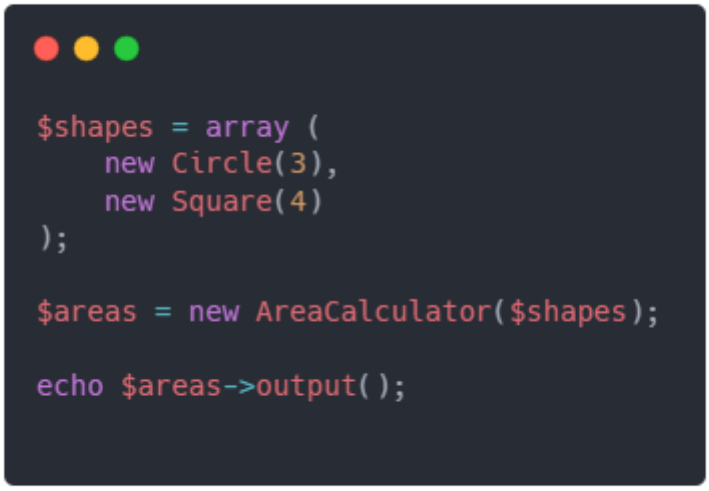
```php
class AreaCalculator
{
    protected $shapes;

    public function __construct($shapes = array())
    {
        $this->shapes = $shapes;
    }

    public function sum()
    {
        // Aquí va la lógica para sumar todas las áreas
    }

    public function output()
    {
        return implode('', array(
            "<h1>",
                "Suma de todas las áreas: ",
                $this->sum(),
            "</h1>"
        ));
    }
}
```

Para utilizar la clase AreaCalculator simplemente instanciamos la clase y le pasamos un *array* con las figuras, mostrando el *output* al final:



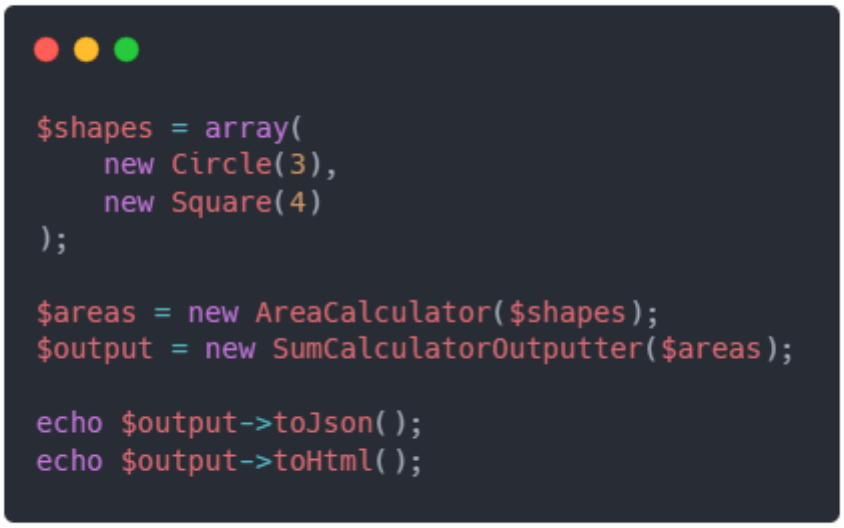
```php
$shapes = array (
    new Circle(3),
    new Square(4)
);

$areas = new AreaCalculator($shapes);

echo $areas->output();
```

El problema del método output es que la clase AreaCalculator además de calcular las áreas maneja la lógica de la salida de los datos. El problema surge cuando queremos mostrar los datos en otros formatos como json, por ejemplo.

El principio Single responsibility determinaría en este caso que AreaCalculator sólo calculase el área, y que la funcionalidad de la salida de los datos se produjera en otra entidad. Para ello podemos crear la clase SumCalculatorOutputter, que determinará cómo mostraremos los datos de las figuras. Con esta clase el código quedaría así:



```php
$shapes = array(
    new Circle(3),
    new Square(4)
);

$areas = new AreaCalculator($shapes);
$output = new SumCalculatorOutputter($areas);

echo $output->toJson();
echo $output->toHtml();
```

Principio de abierto/cerrado (Open/closed principle): Una clase está cerrada a la modificación pero abierta para extenderla. Es decir, la clase abstracta o interfaz queda cerrada a la modificación pero está abierta a su extensión o implementación de la interfaz.

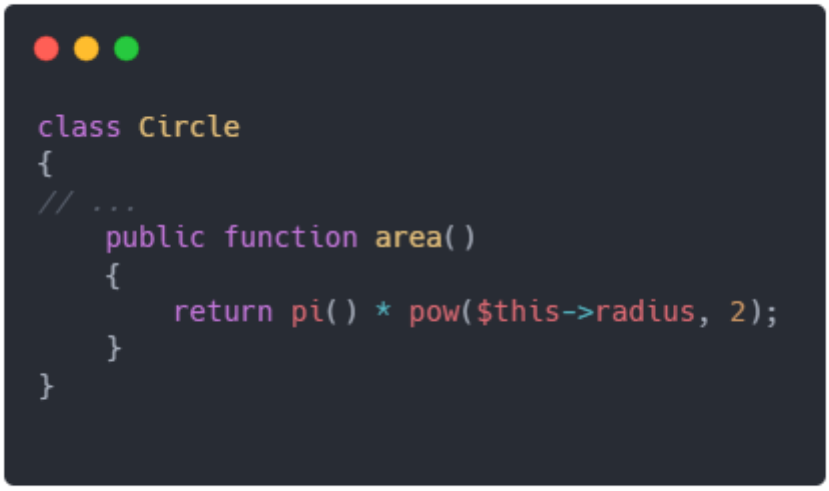
Vamos a ver ahora el método sum de la clase AreaCalculator:

```php
public function sum()
{
    foreach ($this->shapes as $shape) {
        if(is_a($shape, 'Square')){
            $area[] = pow($shape->length, 2);
        } elseif (is_a($shape, 'Circle')){
            $area[] = pi() * pow($shape->radius, 2);
        }
    }
    return array_sum($area);
}
```

Si quisiéramos que el método sum pudiera calcular la suma de más figuras, tendríamos que seguir añadiendo bloques if/else, lo que va en contra del principio Open/Closed. Una forma de hacer este método sum mejor es moviendo la lógica de calcular el área a la clase de cada figura, añadiendo un método area() en cada clase:

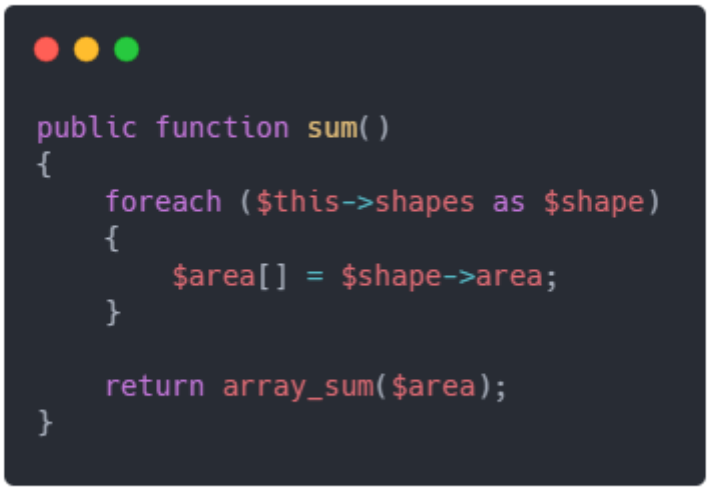
```php
class Square
{
    // ...
    public function area()
    {
        return pow($this->length, 2);
    }
}
```

Lo mismo se hará en la clase Circle:



```php
class Circle
{
// ...
    public function area()
    {
        return pi() * pow($this->radius, 2);
    }
}
```

Ahora para calcular la suma de las figuras proporcionadas dejaremos el método sum de la siguiente forma:

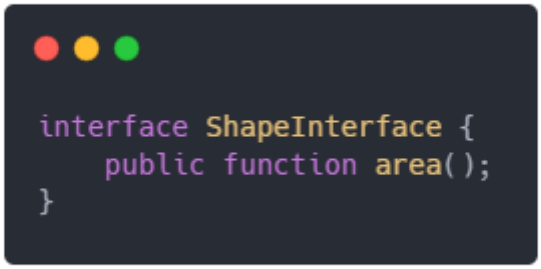


```php
public function sum()
{
    foreach ($this->shapes as $shape)
    {
        $area[] = $shape->area;
    }

    return array_sum($area);
}
```

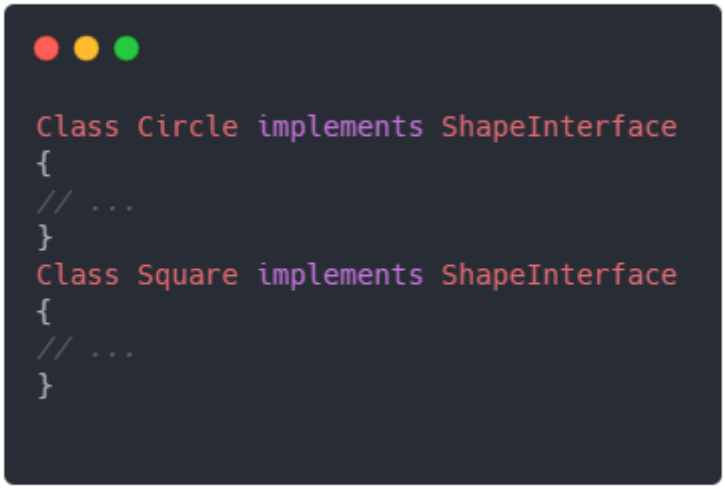
Ahora podemos crear cualquier otra figura y pasarla para calcular la suma que no se romperá el código. Ahora la pregunta es la siguiente: ¿cómo sabemos que el objeto que se pasa a AreaCalculator es realmente una figura o si la figura tiene un método llamado área?

Crear *interfaces* es una parte integral de los principios SOLID. Vamos a crear una *interface* que ha de implementar cada figura:



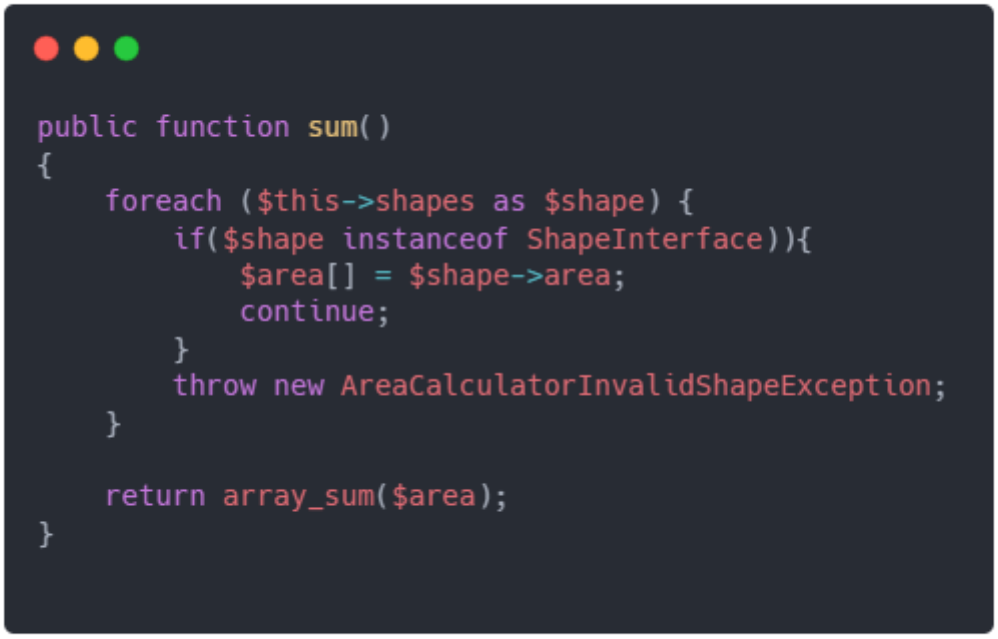
```
interface ShapeInterface {
    public function area();
}
```

Ahora todas las figuras deberán implementarla:



```
Class Circle implements ShapeInterface
{
// ...
}
Class Square implements ShapeInterface
{
// ...
}
```

En el método sum de AreaCalculator podemos comprobar si las figuras proporcionadas son realmente i nstancias de ShapeInterface, y sino, lanzar una excepción:



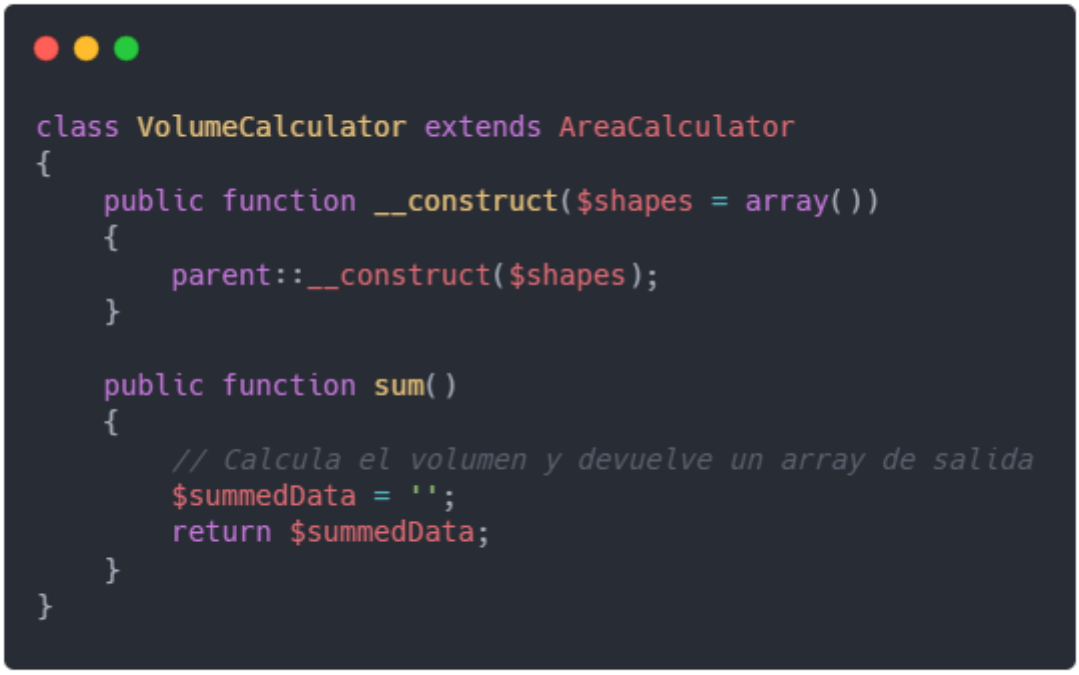
```php
public function sum()
{
    foreach ($this->shapes as $shape) {
        if($shape instanceof ShapeInterface){
            $area[] = $shape->area;
            continue;
        }
        throw new AreaCalculatorInvalidShapeException;
    }

    return array_sum($area);
}
```

Principio de sustitución de Liskov (Liskov substitution principle): Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas. Es decir que si una clase es un subtipo de otra, los objetos de esta clase subtipo podrían sustituir a los objetos de la clase padre sin que el programa sufriera ningún cambio de comportamiento. En otras palabras: si S es una subclase de T, entonces los objetos de T podrían ser substituidos por objetos del tipo S sin alterar las propiedades del problema.

Lo que quiere decir es que cualquier subclase debería poder ser sustituible por la clase padre.

Continuando con la clase AreaCalculator, ahora tenemos una clase VolumeCalculator que extiende la cl ase AreaCalculator:



```php
class VolumeCalculator extends AreaCalculator
{
    public function __construct($shapes = array())
    {
        parent::__construct($shapes);
    }

    public function sum()
    {
        // Calcula el volumen y devuelve un array de salida
        $summedData = '';
        return $summedData;
    }
}
```

VolumeCalculator se podría sustituir por AreaCalculator.

La clase SumCalculatorOutputter quedará:

```php
class SumCalculatorOutputter {

    protected $calculator;

    public function __construct(AreaCalculator $calculator)
    {
        $this->calculator = $calculator;
    }

    public function toJson()
    {
        $data = array (
          'sum' => $this->calculator->sum()
        );

        return json_encode($data);
    }

    public function toHtml()
    {
        return implode('', array(
            '<h1>',
                'Suma de las áreas de las figuras: ',
                $this->calculator->sum(),
            '</h1>'
        ));
    }
}
```

Principio de segregación de la interfaz (*Interface* segregation principle): Dice que los clientes de un programa dado sólo deberían conocer de éste aquellos métodos que realmente usan. Para esto no se deben hacer *interfaces* de propósito general, si esto ocurre habrá que dividirlas en varias *interfaces* para cada propósito específico. Este principio lo que nos facilita es tener un sistema desacoplado de los sistemas de los que depende, de forma que sea más fácil modificarlo, redesplegarlo.

Una clase nunca debe ser forzada a implementar una *interface* que no usa, empleando métodos que no tiene por qué usar.

De nuevo en el ejemplo de figuras, sabemos que también tenemos figuras con volumen, por lo que podr íamos añadir el método volume en la *interface* ShapeInterface:

```php
interface ShapeInterface {
    public function area();
    public function volume();
}
```

Cualquier figura que creemos debe implementar el método volume, pero esto no es lo que queremos ya que fuerza a las figuras sin volumen a aplicar este método. Para solucionarlo podríamos crear una nueva i nterface SolidShapeInterface:

```php
interface ShapeInterface
{
    public function area();
}

interface SolidShapeInterface
{
    public function volume();
}

class Cube implements ShapeInterface, SolidShapeInterface
{
    public function area()
    {
        // Calcula la superficie del cubo
    }

    public function volume()
    {
        // Calcula el volumen del cubo
    }
}
```

Esta forma es mejor, pero a la hora de hacer *type hinting* habría que elegir entre ShapeInterface o Solid ShapeInterface.

Para solucionar esto podemos crear otra interface, ManageShapeInterface, e implementarla en las figur as con y sin volumen, de esta forma podés ver fácilmente que tiene un API para administrar las figuras:

```php
interface ManageShapeInterface {
    public function calculate();
}

class Square implements ShapeInterface, ManageShapeInterface {
    public function area() { /*Hacer cálculos*/ }

    public function calculate() {
        return $this->area();
    }
}

class Cube implements ShapeInterface, SolidShapeInterface, ManageShapeInterface {
    public function area() { /*Hacer cálculos*/ }
    public function volume() { /*Hacer cálculos*/ }

    public function calculate() {
        return $this->area() + $this->volume();
    }
}
```

Ahora en la clase AreaCalculator podemos reemplazar la llamada al método area por calculate y compr obar si el objeto es una instancia de ManageShapeInterface y no de ShapeInterface.

Principio de inversión de la dependencia (Dependency inversion principle): Dice que se debe depender de abstracciones, no de implementaciones. Se le llama así porque se invierte el flujo tradicional de programación mientras que tradicionalmente era el programador que de forma secuencial decidía cuándo se iban instanciando las clases. Ahora lo que ocurre es que determinados sucesos provocan que a través de una entidad externa, en este caso un *framework*, se lleven a cabo las acciones de control necesarias para responder estos sucesos.

Cambiamos ahora el ejemplo por uno relacionado con bases de datos:

```php
class PasswordReminder
{
    private $dbConnection;

    public function __construct(MySQLConnection $dbConnection)
    {
        $this->dbConnection = $dbConnection;
    }
}
```

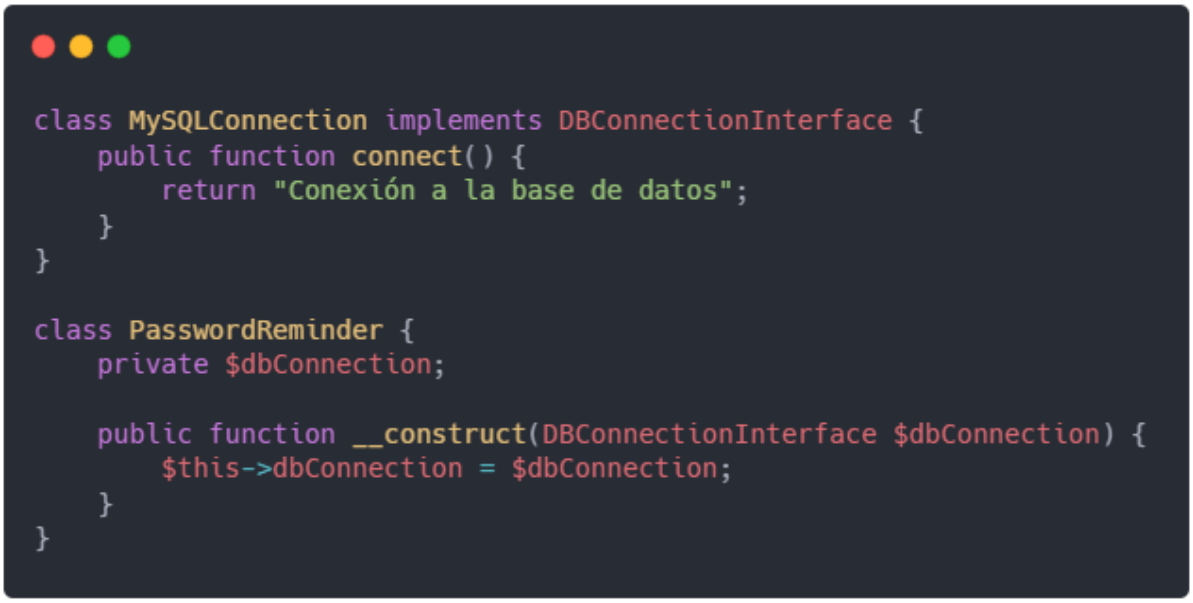
MySQLConnection es el módulo de bajo nivel, mientras que PasswordReminder es de alto nivel. Este ej emplo no respeta el principio SOLID de dependency inversion ya que se está forzando a la clase Passwor dReminder a depender en la clase MySQLConnection.

Si después querés cambiar el motor de base de datos tendrás qué cambiar la clase PasswordReminder también, lo que viola el principio open-closed.

A la clase PasswordReminder no debería importarle que base de datos emplea tu aplicación, y para sol ucionarlo empleamos una interface:

```php
interface DBConnectionInterface
{
    public function connect();
}
```

La interface tiene un método connect y la clase MySQLConnection implementa esta interface. En lugar de hacer type hinting con la clase MySQLConnection en PasswordReminder, lo hacemos con la interface, de forma que no importa el tipo de base de datos que empleemos, que PasswordReminder conectará a la base de datos sin problemas:



```php
class MySQLConnection implements DBConnectionInterface {
    public function connect() {
        return "Conexión a la base de datos";
    }
}

class PasswordReminder {
    private $dbConnection;

    public function __construct(DBConnectionInterface $dbConnection) {
        $this->dbConnection = $dbConnection;
    }
}
```

Ahora podemos ver que tanto los niveles altos como los bajos dependen de abstracciones.

YAGNI

El principio YAGNI nos dice que evitemos desarrollar “por si lo necesitamos después”. Pasa muy seguido que como dev desarrollamos una funcionalidad para cubrir algo que nunca se da y YAGNI viene a solventar este problema, proponiendo hacer exclusivamente lo solicitado y necesario. Tiene su origen en la programación extrema[27], donde prima la simplicidad y donde se opta por no invertir tiempo en dar solución a necesidades todavía inexistentes.

No te olvides de la seguridad

Una práctica super necesaria en el desarrollo es intentar hacer las cosas lo más seguras posible, buscando vulnerabilidades y aprendiendo cuestiones de seguridad de los lenguajes y *frameworks* que estés utilizando. Es un gran plus y no es necesario especializarse en Seguridad Informática si no querés hacerlo, pero sí seguir algunas reglas mínimas, teniendo en cuenta que de alguna forma vas a ser guardián de la información que estés manejando, y que en los últimos años y por diferentes motivos, como el uso de monedas virtuales como Bitcoin o la pandemia mundial COVID-19, los ciber ataques vienen en aumento y son cada vez más difíciles de mitigar. Siempre existieron los ataques, pero con el auge de apps y software en la nube, se multiplicaron los blancos, y si agregamos que cada vez tenemos más

[27] La programación extrema o eXtreme Programming (en adelante, XP) es una metodología de desarrollo de la ingeniería de software formulada por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999).

Al igual que estos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Los defensores de la XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo con lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software. https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema

desarrolladores que quieren sacar soluciones al mercado de forma rápida, muchas veces el resultado es que a esas soluciones no se les dedicó el tiempo suficiente de investigación y desarrollo para prevenir diferentes cuestiones técnicas relacionadas a la seguridad informática.

APRENDÉR A APRENDÉR (y a buscar)

Lo primero que tengo para aconsejarte en este punto y de las cosas más importantes en la carrera de devs que no tengan inglés como idioma madre es aprendérlo. En mi opinión es un diferencial gigante a la hora de programar -ya que los lenguajes que usamos a diario son en inglés-, a la hora de interactuar con otras personas en otros países -siendo que el inglés es el idioma que por defecto va a hablar la mayoría de *devs* del mundo-, al momento de buscar nuevas oportunidades laborales, ya que hoy la externalización de empresas es moneda corriente en el sector IT y por lo tanto no sería raro que termines codeando para una empresa con base en una ciudad de Argentina pero que venda servicios a USA y por último, para hacer siempre mejores sistemas y no usar mezcla de idiomas en tu código fuente (algo que va a pasarte muy seguido). Animate a hablarlo, a no usar tanto el traductor, a escribir mails en ese idioma, usalo como idioma principal en tu sistema operativo y en las cuentas que tengas adquiridas en la nube, etc.

Después, lo que tenés que tratar de mejorar siempre es la forma en que buscás recursos en la web. En el capítulo anterior hablamos de no reinventar la rueda, sino modificarla a tu medida. Para este punto es clave que aprendas a hacer búsquedas rápidas, haciendo un ojeo rápido de la web a donde estés viendo posibles soluciones (Stackoverflow casi seguro).

También está bueno pasar la idea a un papel, haciendo énfasis en cada punto y haciendo anotaciones genéricas que después puedas replicar para otra solución (sí, vas a estar documentando un circuito o proceso).

No tengas miedo de usar el inglés en tus búsquedas en internet, creeme que las respuestas están ahí, esperando a ser encontradas, pero pocas veces en idioma que no sea inglés. Reducí tus búsquedas con palabras sencillas y específicas al problema que estás teniendo. Por ejemplo:

- ~~No puedo conectarme con la base de datos mysql desde visual workbench~~
- [Error code] [Mysql] [Workbench]

- ~~Error en vscode con php cs namespaces~~
- [Each class must be in a namespace of at least one level (a top-level vendor name)] [php-cs]

RECURSOS

En su libro “En sólo 20 horas”[28], Josh Kaufman cuenta algunos tips para aprendér una nueva habilidad o destreza en un periodo corto de tiempo, estableciendo algunas formas de medirnos y teniendo un entorno preparado. Lo que me pareció más importante es que a medida que disminuyen las barreras (falta de herramientas, distracciones ambientales, bloqueos emocionales), aumenta la velocidad de aprendizaje y si aumenta la velocidad de aprendizaje, aumenta tu utilidad en el equipo (o en lo que sea que hagas).

En base a esta idea, y como sección final del capítulo, voy a enumerar algunos ítems que considero muy importantes a la hora de empezar a rendir en un equipo de desarrollo. Si bien al ser Junior vas a poder hacer hincapié en lo que sea que necesites en diferentes momentos, no está de más que empieces a tener una noción básica.

- Git.
- SCRUM.
- Design Thinking.
- XP.
- Kan Ban.
- CI/CD.
- BD Relacionales.
- HTTP.
- CORS.
- Debugear.
- Línea de comandos.
- HTML/CSS/JAVASCRIPT.
- Responsive Design.
- Herramientas de Desarrollo en el Explorador (como Chrome Dev Tools).

[28] https://www.amazon.com/-/es/Josh-Kaufman-ebook/dp/B00KY7B3PK

PALABRAS FINALES

ABRAZAR EL CAMBIO

Si decidís que es por acá, vas a tener que acostumbrarte al cambio, abrazarlo y tratar de aprovechar lo que cada etapa y cada situación te devuelve. El mundo del desarrollo es, entre muchas otras cosas, dinámico. Lo que se conoce hoy como agilidad no es más ni menos -de modo simple y para que lo entiendas- que un conjunto de valores que hacen que una persona y un equipo sea más o menos flexible. La transparencia, el respeto, el foco, y la apertura al cambio son algunas de las habilidades que vas a tener que trabajar para mejorar tu *performance*.

En el mundo del desarrollo es común encontrarnos con situaciones en donde vemos cómo las personas se rehúsan al cambio casi de forma automática. Ya sea por temor a perder su trabajo, a empezar a trabajar de forma diferente, a estar más controlados (aunque deberían entender, si ese es su miedo, que lo que se busca es la mejora continua y no el control limitante).

El temor al cambio se puede ver en diferentes planos, ya sea en el tecnológico, cuando los y las líderes se rehúsan a cambiar la tecnología que conocen por miedo a no poder entregar más resultados como lo vienen haciendo, o a quedar relegados por esa otra persona que conoce la tecnología mucho mejor, también en un plano laboral/personal, cuando a veces tenemos miedo de dar el salto necesario para cambiar de trabajo y avanzar en nuestra carrera.

Si ya sos parte de este mundo y tuviste posibilidad de trabajar en una startup y una corpo, seguro notaste la diferencia abismal entre las dos estructuras. La última, más grande, jerárquica y rígida, resulta en que los cambios sean más lentos y las transformaciones digitales no sean tales, sino cambios impuestos. Igual los gobiernos, que no dejan de ser grandes empresas rígidas, aunque hay que reconocer que en los últimos años están haciendo algunos esfuerzos para intentar sumarse a la ola 4.0 (aunque lamentablemente en algunos casos, sea una cuestión de voto-bait).

Personalmente tuve una experiencia de resistencia al cambio muy esclarecedora, en donde teníamos como misión desarrollar e implementar un software de gestión en un área de una de las empresas en donde trabajé. Los usuarios se resistieron desde el día cero, pensando que lo único que buscaba el sistema era saber qué hacía cada persona en cada minuto de la jornada laboral: si iban al baño, si se pasaban 5 minutos de la hora programada para desayunar o si las tareas encomendadas les llevaba 20 minutos más o menos. De más esta aclarar que ese no era el motivo por el cual estábamos trabajando tan duro. A nadie le importan esos números, necesitábamos información real para mejorar el área, hacerla más rentable y que pueda darle un valor agregado a la empresa. Con el tiempo, y debido a varios factores, entre el o los más importantes el no contar con información y no poder decidir, el área terminó por cerrarse. Hoy en día me pregunto qué hubiera pasado si hubiéramos tenido datos.

NO TE COMPARES

Es normal cuando empezamos en algo nuevo que pongamos la vara lo más alta posible y soñemos con ser los próximos Elon Musk[29], Mark Zuckerberg[30] o Susan Wojcicki[31]. Queremos vernos en lo más alto, exitosos y exitosas, llenos de poder y con la posibilidad de liderar grandes empresas. En diferentes oportunidades tuve la posibilidad de hablar con algunas personas que me preguntan "¿Qué consejos les darías a los más chicos que están arrancando?". La respuesta, casi de casete, es la misma: No te compares. Las personas que acostumbramos a seguir formaron su camino en base a muchísimo esfuerzo, lágrimas, horas dedicadas y emprendimientos frustrados. No lo consiguieron de un día para el otro, no lo hicieron sin dejar de dedicarle horas a su círculo íntimo, hijos, hijas, madres, padres, esposos y esposas, a sus amistades, y a sí mismas. No creas que esas personas tienen todo resuelto, ni mucho menos. Son justamente personas, con problemas de peso, ansiedad, recaídas, y un montón más de cosas que nos caracterizan a los seres humanos. No busques ser el próximo Elon Musk, buscá hacer algo distinto, algo que a él no se le haya ocurrido hacer. Buscá que las próximas generaciones quieran seguirte a vos también. Y con todo esto no estoy desmereciendo el trabajo ni el aporte de ninguna de esta de estas personas (¿quién pudiera hacer lo que hicieron y hacen, no?), sólo estoy dándote un consejo super valioso, ya sea que estés arrancando una carrera en el deporte -¿de verdad creés que podés ser el próximo Leonel Messi? ¿Y si te esforzás en ser útil a tu equipo actual?- o en tecnología. Recordá que las posibilidades son siempre infinitas, y tenés que lograr que tu cabeza sea tu mayor aliada y no tu mayor enemiga.

MEMENTO MORI

Nuestro paso por el Planeta Tierra es efímero. Puede durar algunos años, poco más, poco menos, pero tiene un fin, y a veces antes de lo previsto. Es una realidad difícil de aceptar pero es así.

Para terminar este libro quiero contarte acerca de esta frase: "Memento Mori" -recuerda que morirás-.

Era costumbre hace muchos años, en el desfile victorioso de un general por las calles de Roma, tener tras él un siervo que se encargaba de recordarle las limitaciones de la naturaleza humana, con el fin de impedir que incurriese en la soberbia y pretendiese, a la manera de un dios omnipotente, usar su poder ignorando las limitaciones impuestas por la ley y la costumbre. De esta manera, se intentaba tener a los generales "con los pies sobre la tierra".

Sé muy bien que el comienzo de esta hermosa profesión es agotadora, llena de dudas, con muchas ganas de demostrar que somos valiosos y que podemos hacer cosas geniales, y a veces eso nos termina jugando una mala pasada y el entorno nos termina absorbiendo. Nos volvemos codiciosos en vez de ambiciosos, cerrados en lugar de abiertos, queremos proteger lo nuestro y nos olvidamos de todo y todos los demás. Está en nosotros ser partícipes de nuestra historia y disfrutar el camino. Esta profesión depende muchísimo de tu estado de ánimo, de tus ganas de hacer, de tus ganas de interactuar con otras personas. No la concibo de otra forma y el código que hacemos termina siendo resultado de lo bien o lo mal que estemos.

Usá tu Memento Mori, ya sea tu familia, tus mascotas, tus amistades o los pequeños placeres del día a día para recordarte a vos mismo que la vida es corta, y por sobre todo, ¡hay que disfrutarla!

---

[29] Elon Reeve Musk (Pretoria, Sudáfrica; 28 de junio de 1971) es un físico, emprendedor, inventor y magnate sudafricano, nacionalizado canadiense y estadounidense. Cofundador de PayPal, SpaceX, Hyperloop, SolarCity, The Boring Company, Neuralink y OpenAI. Es director general de SpaceX, de Tesla Motors, presidente de SolarCity y copresidente de OpenAI https://es.wikipedia.org/wiki/Elon_Musk

[30] Mark Elliot Zuckerberg (White Plains, Nueva York; 14 de mayo de 1984) es un programador y empresario estadounidense, uno de los creadores y fundadores de Facebook. https://es.wikipedia.org/wiki/Mark_Zuckerberg

[31] Directora ejecutiva de YouTube. Es graduada con honores en Historia y Literatura en la Universidad de Harvard, además posee una maestría en economía por la Universidad de California en Santa Cruz, y un MBA de la UCLA. https://es.wikipedia.org/wiki/Susan_Wojcicki