El programador Mediocre

Craig Maloney

El programador mediocre

Craig Maloney

Victorhck

Traducido por Victorhck

El programador mediocre

El programador mediocre

Introducción

0.1 ¿El programador mediocre?

Afrontémoslo: no queremos ser programadores mediocres. ¡Queremos ser programadores `[geniales,increíbles,superlativos]`! Queremos ser los programadores a los que llaman cada vez que están en un aprieto, y queremos ser los programadores que se sumergen en bases de código desconocidas y producen código perfecto en cuestión de minutos. Código que estaría en el Louvre como una obra de arte, estudiado por generaciones de programadores por su belleza intrínseca y funcionalidad excepcional.

¿Por qué íbamos a querer ser programadores mediocres? ¿No es mediocre lo opuesto a genial? ¿No deberíamos esforzarnos por ser grandes programadores?

Por supuesto, deberíamos esforzarnos por ser grandes programadores a largo plazo, pero para llegar a ser grandes programadores primero debemos pasar por la etapa de ser programadores mediocres.

Los programadores mediocres saben que no son (todavía) grandes programadores. Los programadores mediocres ven la distancia entre donde se encuentran actualmente y la grandeza que quieren en sus trabajos de programador. Son conscientes del trabajo que implica ser un gran programador y creen que si hacen el trabajo también llegarán a ser grandes programadores.

Pero también ven sus propias faltas y carencias. Ven el historial de su navegador lleno de búsquedas en internet de sintaxis y conceptos básicos. Ven sus archivos de correo electrónico de preguntas que han hecho a otros programadores. Se estremecen ante su código de hace varios meses y se preguntan si alguna vez llegarán a ser grandes programadores con todos estos errores y traspiés. Ven la brecha entre ellos y los grandes

programadores, y se siente como si la brecha se ensanchara a cada paso del camino.

Los programadores mediocres se preguntan si vale la pena. Se preguntan si deberían hacer algo más con sus vidas además de programar computadoras. Tal vez no sean tan buenos como creían, o tal vez les falte ese talento especial que tienen los grandes programadores. Tal vez sientan que aprendieron cosas equivocadas al principio de sus viajes, o tal vez piensen que deberían haber comenzado antes.

Ven que otras personas que tienen un gran éxito como programadores y se preguntan si estuvieron ausentes el día en que se entregaron los grandes genes programadores.

La verdad es que todos somos programadores mediocres de alguna forma. Seguimos realizando preguntas y teniendo que echar un vistazo a sintaxis y conceptos en nuestro día a día cuando programamos. Las computadoras continúan evolucionando y los programadores continúan añadiendo complejidad cada día a las tareas de programación. Conlleva mucho ancho de banda mental mantener todos esos conceptos frescos en nuestra mente.

0.2 ¿Por qué este libro?

Este libro trata de ayudarte en tu viaje como programador mediocre. Juntos descubriremos algunos conceptos erróneos comunes que tenemos sobre la programación, el fracaso y el crecimiento. Entenderemos que el acto de programar y desarrollar es algo que hacemos todos los días. Todos los días podemos mejorar en pequeñas cosas. Son estos pequeños cambios los que nos transforman de programadores mediocres en mejores programadores.

Hay muchos libros sobre cómo convertirse en un mejor programador. Tienden a tener listas de verificación y otros consejos que el autor considera lo suficientemente importantes como para que los hagas a fin de convertirte en un mejor programador. Suelen centrarse en mejoras específicas, como por ejemplo elegir el mejor editor, escribir mejores casos de prueba o beber mucha agua. Esos libros tienen muchos consejos útiles, pero se leen como

una lista de cosas que debes hacer todas a la vez para tener éxito. Este libro intentará no cargarte con más trabajo (probablemente ya tengas suficiente). Más bien, discutiremos lo que se siente siendo un programador. Hablaremos de las emociones que aparecen mientras programamos, los sentimientos de frustración, culpa, ira e insuficiencia. Trataremos de las dificultades para aprender cosas nuevas y mantener tus habilidades actualizadas. Hablaremos de esos momentos en los que tienes ganas de darte por vencido y alejarte de la informática y si esos sentimientos provienen de un lugar de amor o de una preocupación por no está al día.

Este libro es un viaje personal para ambos. Es una memoria de mi tiempo como programador y mis sentimientos a lo largo del camino. He pensado muchas veces en rendirme y encontrar una carrera diferente, pero hacer otra cosa que no sea programador de computadoras me asusta aún más. ¿Significa eso que estoy atrapado en un uróboro (*N. del traductor: un símbolo que muestra a un animal con forma de serpiente que engulle su propia cola y que forma un círculo con su cuerpo. El uróboro simboliza el ciclo eterno de las cosas*) perverso de autocompasión y dudas? Difícilmente. Significa que necesito profundizar más para comprender por qué elegí el camino de ser programador y darme cuenta de que me costó mucho llegar aquí y que me llevará mucho más llegar a donde quiero estar. Es un compromiso de ver las cosas como son ahora y seguir adelante desde donde estoy parado.

0.3 Descargo de responsabilidades

No soy doctor ni terapeuta. No tengo cualificación para darte un consejo médico. Soy programador. Toda la información de este libro está dada desde la perspectiva de un programador con dificultades y no debe tomarse como un consejo médico. Si necesitas ayuda de un profesional médico, por favor busca a una persona que pueda ayudarte. (Hay un capítulo entero sobre buscar ayuda de otras personas casi al final de este libro).

Comencemos nuestro viaje averiguando dónde estamos y recordando qué nos llevó a este lugar.

0.4 De esta traducción

No recuerdo cómo conocí el texto de Craig Maloney, quizás dí con él porque le sigo en Mastodon y desde ahí conocí su proyecto. Cuando lo conocí, antes de leerlo, quizás pensé que sería un montón de recursos y consejos técnicos para las nuevas personas que llegaran a esto de la programación, pero me resultó curioso el título.

Después de leerlo, me dí cuenta que no. Que no se hablaba de nada técnico sobre programación, que lo que aquí se cuenta tiene que ver más con el aspecto psicológico y mental en lo referente a la programación que con aspectos técnicos.

Yo, sin ser programador, encontré en el libro algunos consejos y trucos interesantes para aplicar en mi día a día en algunos aspectos de mi faceta digital (mi blog, traducciones, colaboraciones, etc).

Y como siempre, de ese interés personal nació esta traducción que hoy tienes delante, después de pedir el correspondiente permiso al autor original. Me gustó lo que se contaba y la traducción “me obligaba” a tener que leer el texto y comprenderlo para realizarla y poder compartirla con todo el mundo.

Si tu trabajo tiene que ver de alguna manera con la programación y en algún momento te has sentido una persona perdida, frustrada o fuera de lugar, este texto te ayudará a saber que es algo común y el autor compartirá contigo sus visiones de su propio viaje en el mundo de la programación y cómo consiguió ir pasando etapas en ese viaje. Tu tendrás tu propio camino en el viaje de la programación, pero tomar como referencia lo que otras personas han sentido quizás te sirva como ayuda en los momentos bajos del camino, espero que sea así.

Pero si no eres programador, también es un texto interesante, este es un relato en el que el autor revela sus puntos flacos, y comparte algunos de los recursos que le han ayudado a conseguir sus metas. Quizás tu puedas aplicarlas a tu día a día en otras facetas, a mí me han servido.

La traducción la he realizado en un repositorio git hospedado en Codeberg donde tienes disponible todos los capítulos y el código original que utiliza el autor para generar su página web.

Si este u otro contenido que creo te resulta interesante, siempre puedes agradecérmelo mediante una donación en Liberapay. Pero tanto si donas como si no, todo el contenido que creo estará libre para que lo disfrutes.

1 El viaje del programador mediocre

1.1 Cómo hemos llegado aquí

Tienes tu propia historia única de cómo has llegado hasta aquí como programador. Es posible que te hayas enterado de qué era eso de la programación cuando eras un niño curioso que quería ver lo que un ordenador podía hacer. O quizás es posible que hayas llegado como un adulto que escuchó sobre estas cosas llamadas computadoras que puedes programar. Sea cual sea tu caso, has tenido un viaje para llegar hasta este punto y aprendiste una cierta cantidad de cosas para llegar hasta aquí. Pasaste tu tiempo libre aprendiendo a escribir código, o tuviste la suerte de poder trabajar en la programación como parte de tu trabajo. Fuiste a la escuela para aprender más sobre programación o quizás tomaste clases de extras. Compraste libros o leíste artículos en Internet para aprender más sobre programación. Cualquiera que sea el camino que hayas tomado, iniciaste tu viaje como programador.

Y ahora te sientes estancado.

Miras a tu alrededor y te preguntas si alguna vez sabrás todo lo que deberías saber. Lees un artículo en un sitio y se despierta tu interés. Un amigo en Internet menciona esta cosa genial que ha encontrado y espera que aprendas más al respecto. Tu colega encontró algo que podría resolver un problema que tiene en un proyecto y ahora tienes algo más que aprender.

Casi todas las semanas surgen nuevos temas y tecnologías. Estas “cosas” se van introduciendo en nuestras discusiones de programación y en nuestro trabajo. Quizás encuentres estas cosas nuevas que aparecen en anuncios de trabajo que requieren un mínimo de más de 3 años de experiencia, y te preguntas cómo alguien podría tener ese nivel de experiencia. Tal vez elegiste ignorar estas cosas durante un tiempo y ahora se han convertido en

un factor determinante en tu trabajo. Es como si alguien cambiara un simple bit de estado 0 a 1 y ahora ya no eres digno de ser llamado programador a menos que hubieras sido uno de los primeros en adoptar estas cosas.

Cada una de estas experiencias te hace sentir como si estuvieras incompleto sin aprender estas cosas nuevas. Nos muestran que no importa cuál sea nuestra experiencia actual, todavía hay lagunas en nuestro conocimiento que deben ser llenadas. Cuando miras hacia el horizonte, puedes ver las brechas que surgen entre el lugar donde estás y el lugar donde crees que deberías estar.

1.2 La brecha

Elegí la palabra “brecha” (*Nota del traductor: en inglés usa la palabra “gap” que tiene distintos significados como brecha o hueco*) para describir la diferencia entre dónde estás ahora y dónde crees que deberías estar por una buena razón. Una brecha es algo impuesto por otros, ya sea por la fuerza o por negligencia. Si aparece un agujero en la cerca de tu jardín, significa que la cerca es más débil para proteger el jardín. Una brecha también puede ser algo que requiere nuestra atención. En inglés la expresión “Mind the gap” es una frase acuñada a finales de la década de 1960 por el metro de Londres para advertir a la gente sobre el espacio que hay entre la plataforma y los vagones del tren. Si las personas no tienen cuidado con ese espacio, podría generar una situación insegura.

La brecha en este caso es la distancia entre nuestras habilidades actuales y dónde pensamos que deberíamos estar. A veces las brechas son auto impuestas debido a nuestros propios deseos de mejorar, pero la mayoría de las veces las brechas son impuestas de manera externa.

Uno de los grandes creadores de brechas en nuestra profesión como programadores es el cambio. Como programadores estamos siempre alerta del ciclo de los cambios en la cultura de la programación. Estamos constantemente enfrentándonos a los cambios que nos llegan: cambios de tecnología, cambios de prioridades en el trabajo o incluso cambios en

nuestra estrategia para tratar de mantenernos al día con las demandas que se nos piden.

El cambio también puede provenir de dentro de nuestra comunidad. En nuestra comunidad de programadores y usuarios puede cambiar a nuevos puestos de trabajo o nuevas tecnologías. Es posible que ya no obtengamos el apoyo que necesitamos para hacer nuestro trabajo y que nos enfrentemos a la posibilidad de que nosotros también necesitemos actualizar nuestras habilidades o nos quedemos atrás en una comunidad abandonada.

El cambio puede provocar estrés. Es común en el círculo de los programadores porque las cosas cambian a menudo. Lo que funcionaba el viernes por la tarde puede que ya no sirva el lunes por la mañana debido a un cambio en una biblioteca que estábamos utilizando. Nuestro entorno de desarrollo podría romperse debido a una actualización que no se aplicó de la manera correcta. El código en producción podría necesitar algunos parches de seguridad y esos parches significan que tenemos que volver a hacer una gran parte de nuestro software porque si no ya no funcionaría. Hay una gran variedad de maneras en las que estamos inmersos en ese ciclo del cambio.

No todo el cambio es malo. El software que utilizamos podría tener razones muy válidas para cambiar. Las nuevas funcionalidades que deben ser añadidas requieren nuevas formas de pensar nuestro código. Las correcciones a problemas de seguridad podrían significar que nuestros sistemas ahora serán más resistentes frente a ataques externos, pero requieren diferentes formas de utilizar ese sistema. Nuevas y mejores optimizaciones pueden llevar a que nuestro código se ejecute de una manera más rápida pero necesite algunos ajustes para que pueda beneficiarse de esa velocidad. Refactorizar una API puede llevarnos a interactuar con nuestros sistemas de una manera más limpia y concisa, pero puede introducir cambios que no son compatibles con el código actual.

El cambio puede ser positivo, pero el cambio requiere tiempo y esfuerzo para adaptarse a él. La brecha solo se puede cerrar si tenemos los recursos y el tiempo necesario para trabajar en ella. Si estamos luchando con nuestra carga de trabajo actual y alguien cambia la forma en que funciona algo, ahora tenemos que tener en cuenta nuestro tiempo para adaptarnos a ese

cambio. Si nuestro músculo de memoria mental sabía cómo funcionaba algo y ahora se enfrenta a un cambio significativo, requiere que volvamos a entrenar ese músculo de memoria mental para que coincida con la forma en que funciona ahora. Y si ya sientes que no comprendes los sistemas y entornos en los que estás trabajando, añadir cambios adicionales pueden dejarte varado entre las brechas recién formadas de tu conocimiento.

1.3 Cerrando la brecha

Me encantaría decirte que hay una manera de cerrar esa brecha, decirte que has aprendido todo y que ya te puedes sentir a salvo porque ya dominas la totalidad de la programación.

Desafortunadamente, yo no he encontrado una manera de cerrar las brechas.

Puedes seguir aprendiendo todo lo que hay que saber sobre cualquier tema sobre el que hayas escogido aprender. Puedes realizar todos los cursos disponibles. Puedes asistir a todas las charlas y debates, leer informes que traten del tema e incluso realizar tu propia investigación y aún así te puedes seguir sintiendo como que no has cerrado definitivamente esa brecha.

Así que, si no hay una manera de cerrar por completo la brecha ¿Qué puedes hacer?

Tienes tres posibles opciones disponibles:

La primera opción es no intentarlo. Asumir que siempre habrá algo más que aprender. ¿Por qué preocuparse? Es más sencillo auto convencerte de que nunca serás capaz de cerrar esa brecha del conocimiento. La mejor opción es, te dices a ti mismo, quedarte con lo que sabes y aguantar todo el tiempo que puedas.

La segunda opción es intentar hacerlo todo a la vez. Te lees cada libro, cada artículo de un blog, cada nuevo informe, vídeo, o lo que sea para intentar aprender sobre el tema. A continuación, te das cuenta que solo tienes una cantidad finita de tiempo para aprender sobre el tema y que no puedes consultar todo ese material a la vez. Revisas tu progreso y te desesperas

porque tu aprendizaje no está progresando tan rápido como te gustaría. Culpas a los materiales de aprendizaje por tu falta de progreso y buscas algo más que te ayude a aprender mejor el tema. La frustración se instala cuando te culpas a ti mismo por no haber comenzado antes a cerrar esta brecha.

La tercera opción es el enfoque más mesurado. Empiezas poco a poco con pequeñas tareas y avanzas hacia la meta. En lugar de ver la brecha como algo que se deba cerrar, te das cuenta de que no puedes conocer la totalidad de cualquier tema que estés aprendiendo. Disfrutas del conocimiento que recibes en la búsqueda del aprendizaje del tema. Mantienes un ritmo constante para aprender tanto como puedas. En lugar de lamentarte por no haber comenzado antes, estás agradecido de haberlo hecho.

De estas tres opciones, la primera y la tercera son en las que encontrarás mayor satisfacción. La primera opción (no intentar) te permite acomodarte con el conocimiento que tienes. Pero hay una desventaja en simplemente permanecer en el lugar. Nuestra industria está en constante cambio y la tecnología sigue avanzando. Lo que solía ser la norma se convierte en algo ya obsoleto y lo que estaba a la vuelta de la esquina se convierte en lo que está siendo demandado.

Una de las habilidades más útiles de un programador es la capacidad de adaptarse a las nuevas tecnologías. A medida que cambia nuestro entorno tecnológico, nuestra capacidad para adaptarnos a esos cambios nos permite continuar como programadores. Máquinas más rápidas, diferentes tecnologías, diferentes dispositivos, diferentes requisitos, cada uno de estos nos trae desafíos emocionantes si los reconocemos. Pero también consumen tiempo para aprender y crean brechas en nuestro conocimiento. Confiar en nuestro conocimiento previo para llevarnos a través de estos cambios no será suficiente. Tenemos el desafío de adaptarnos al nuevo entorno.

La segunda opción (tratar de acapararlo todo y frustrarse) es el camino menos óptimo. Tratar de aprender con todos los recursos disponibles e incrustándolos en nuestros cerebros es un camino hacia la frustración, la fatiga y el agotamiento. Muchos desarrolladores intentan esto porque sienten la necesidad de adaptarse al nuevo entorno, pero es difícil hacer muchos cambios radicales de una sola vez. Es como tratar de desarrollar alas para volar porque llegas tarde a una cita: te sentirás frustrado por su

incapacidad para desarrollar alas y aun así llegarás tarde a tu cita. La segunda opción también mide tu progreso en función de cuánto más crees que debes avanzar. Descarta el progreso hecho y crea un ciclo sin fin hacia una línea de meta en continuo movimiento.

De las tres opciones, la tercera opción es la que tiene más sentido. El tomar una decisión mesurada para cerrar las brechas de nuestro conocimiento nos permite disfrutar más del proceso de aprendizaje. Al conseguir pequeños pasos en nuestro viaje, esto nos concede pequeñas victorias en el camino recorrido. En vez de esperar una gran transformación nos permitimos cambios graduales para adaptarnos a nuestro entorno.

Con este acercamiento mesurado y gradual también obtenemos la sabiduría de que no tenemos que cerrar todas las brechas. Nos permitimos seguir aprendiendo en las áreas que necesitamos y de forma gradual vamos construyendo nuestras propias habilidades.

También nos podemos dar cuenta, que cerrar la brecha por completo es una ilusión. Mientras seamos personas vivas siempre habrá brechas, ya que surgen cosas nuevas cada día. Podemos escoger cómo de expertos podemos ser en cualquier tema que hayamos escogido aprender. Incluso podemos esforzarnos por aprender tanto como podamos, pero lo hacemos con una sensación de alegría en el proceso de aprendizaje, no por una necesidad perversa de tratar de recopilar la totalidad del conocimiento informático en nuestras cabezas.

1.4 El viaje es la recompensa

El secreto para avanzar en nuestro viaje como programadores y desarrolladores es darnos cuenta de que cada paso que damos en ese camino es valioso y digno de nuestra atención. El hecho de que no hayamos aprendido la última tecnología en quince días no significa que debamos dejar de intentarlo. Tampoco aprender una tecnología solo para ver cómo se ve eclipsada por otra cosa, significa que nuestro tiempo de aprendizaje se haya desperdiciado. Cada desafío al que nos enfrentamos, cada tecnología que aprendemos y cada hora que dedicamos a programar es un paso más en

nuestro viaje para convertirnos en mejores programadores. Cada error y giro equivocado nos presenta oportunidades para aprender de esos errores y crecer como programadores y desarrolladores. No existe un camino perfecto para mejorar en esto. Incluso si lo hubiera, estoy seguro de que podríamos señalar cualquier punto de nuestro pasado y decir que fue menos que perfecto. No es posible trazar el camino perfecto desde donde estamos hasta donde deseamos estar. Peor aún, es una ilusión que nos impide avanzar en nuestra práctica diaria de programación y desarrollo. Puede parecer trillado decir “el viaje es la recompensa”, pero cada día que trabajamos como programadores y desarrolladores estamos un día más cerca de cerrar esas brechas en nuestro conocimiento y estar más contentos de ver cómo crecen nuestras habilidades.

2 Compañeros de viaje

2.1 Programadores famosos

A lo largo de la historia de la computación ha habido personas que han demostrado increíbles capacidades a la hora de crear código. Residen en el panteón de los grandes programadores informáticos: Ada Lovelace (la primera programadora de computadoras), Dennis Ritchie (el creador del lenguaje de programación C), Rear Admiral Grace Murray Hopper (creadora de COBOL y acreditada por encontrar el primer "error" informático documentado). (*N. del traductor: un error informático en inglés es llamado "bug" que también se puede traducir como bicho o insecto*) y muchos más. También tenemos desarrolladores en nuestras propias comunidades, que tienen un cierto grado de notoriedad, ya sean las personas que escribieron el lenguaje de programación que usamos actualmente, las personas que mantienen el sistema operativo que usamos o alguien que saltó a la fama en nuestra comunidad preferida. Puede ser intimidante cuando nos comparamos con estas grandes personas. Después de todo, cualquier cosa en la que estemos trabajando actualmente podría no estar a la altura de lo que estas personas hayan hecho. Peor aún, podemos estar trabajando en algo similar y sentir que cualquier cosa en la que estemos trabajando nunca estará a la altura de lo que estas personas han logrado. Incluso podemos ser amigos de programadores que parecen resolver las cosas mucho más rápido y de una manera más elegante que nosotros y maravillarnos de cómo parecen tener todo ese conocimiento al alcance de la mano que posiblemente no podamos entender.

2.2 Entre bastidores frente a la actuación final

Uno de los mejores consejos que he recibido sobre compararnos con los demás, es darnos cuenta de que la comparación es entre nuestra vida entre

bastidores *versus* la actuación final que ven los espectadores. La metáfora se basa en el teatro, donde los artistas saben todo lo que no está bien sobre la actuación de su propio grupo de teatro y lo comparan injustamente con la actuación final de otro grupo de teatro. Esta metáfora es útil en nuestro caso porque tendemos a comparar todas las cosas que sabemos (las largas horas de crear código improductivas, las dificultades para aprender, etc.) con el producto final de otra persona. No vemos su lucha para hacer que las cosas funcionen, o sus innumerables horas haciendo prototipos de mierda y proyectos sin terminar antes de hacer lo que admiramos. Permítete tener un lugar entre bastidores desordenado y hacer muchos ensayos, y comprende que se necesita esfuerzo y práctica para realizar una buena actuación.

2.3 El atractivo del *post-mortem*

Hay una tradición en algunos proyectos de programación (especialmente en proyectos de desarrollo de juegos donde hay un final claro para el proyecto cuando se envía el producto) de hacer una autopsia del proyecto. Lo que hace la autopsia, es permitir que los desarrolladores de un proyecto indiquen lo que salió bien y lo que no salió bien. Las mejores autopsias tienden a ser relatos sinceros de los éxitos y fracasos de un proyecto.

La autopsia puede ser una mirada fascinante al desarrollo de un proyecto. Me he encontrado leyendo muchas de estas, en busca de información sobre el proceso de desarrollo. Pero hay una trampa sutil en la autopsia: son un recuerdo de eventos desde el punto de vista de un proyecto exitoso (o fallido). Son un recuerdo de alguien que trabajó en un proyecto que fue lo suficientemente exitoso, como para que tu dediques tiempo a leer sobre los altibajos de ese proyecto. Están escritos desde una perspectiva en la que el éxito del proyecto es una conclusión inevitable (o están escritos sobre proyectos que se destacaron por cómo fallaron o no cumplieron con las expectativas de los involucrados). Puede llevar a la creencia de que en lo que estás trabajando no es tan importante como las cosas en las que están trabajando otras personas. Pero no sabemos la importancia de nuestro proyecto en tiempo real. Incluso la gente en la autopsia no sabía si su proyecto funcionaría o tendría éxito mientras trabajaban en él. Es posible que nuestros proyectos nunca vean la luz del día o que sean algo que

cambie el mundo. No podemos saber el valor de aquello en lo que estamos trabajando mientras lo hacemos (aunque podemos tener una idea de si *sentimos* que nuestro trabajo es importante o no).

Una autopsia también tiene el beneficio de la retrospectiva. Las decisiones que eran claras y definitivas en ese momento podrían no tener mucho sentido cuando se las compara con los datos obtenidos más adelante en la vida útil del proyecto. También hay un problema con la "memoria selectiva" en la que es posible que algo no se recuerde con la misma claridad o se combine con otros eventos. Declaraciones confiadas como "Sabíamos que esto no habría funcionado" en realidad podrían haber sido "No estábamos seguros de si esto funcionaría, así que probamos varias cosas. Todas no funcionaron". Debes considerar a cualquiera que escriba sobre su pasado como un narrador poco confiable. Cierto, pueden ser los mejores y más informados narradores que tenemos, pero no tienen una perspectiva objetiva de lo que sea que estaban creando. Tienen sus propios prejuicios y razones para las historias que presentan en una autopsia. Debes tratar estas autopsias como tratarías una autobiografía de una persona famosa: una fuente primaria con una agenda para mostrar el tema de la mejor manera posible.

No hay nada de malo en leer una autopsia sobre un proyecto, podemos aprender mucho sobre cómo se ejecuta un proyecto (o no se debe ejecutar) y qué peligros debemos tener en cuenta si seguimos un camino similar, pero hay que comprender que estás leyendo un informe (ya sea de una persona o de un equipo de personas). Tienen la perspectiva de alguien profundamente involucrado en el conflicto. Estás mirando sus recuerdos de tácticas, no la estrategia general que los trajo al lugar.

2.4 Clasificación de programadores

Hay muchas medidas que la gente usa para clasificar a los programadores. Es probable que hayas visto que estas medidas se manifiestan de diferentes maneras: sitios de competencia, números de *commits* en un proyecto, medidas de productividad, tiempo para entregar el código y los buenos sentimientos viscerales que podamos sentir, como se hacía antiguamente.

Nos lo hacemos a nosotros mismos y a los demás. Comparamos nuestro trabajo con el trabajo de nuestros compañeros y gente que admiramos, pero eso puede llevarnos a hacer comparaciones que no son objetivas o no se basan en todos los datos. Puedo compararme con personas que hacen programación de bajo nivel y encontrar que no estoy a la altura en ese ámbito. No importa que no haya hecho mucha programación de bajo nivel, la comparación es válida. O puedo compararme con personas que fueron asesoradas por programadores cuyos nombres son legendarios en su campo. Encontraré brechas entre mi conocimiento y el de ellos, porque no tuve acceso a esos mentores (o peor aún: no aproveché los mentores a los que podría haber accedido. ¡Ups!). Comparaciones como estas no ayudan y nos llevan a castigarnos por no ser otra persona. Nuestra evaluación de nuestros proyectos e historia nos da la conclusión de que no somos esa otra persona, ni podríamos ser nunca esa otra persona.

El mayor problema con la clasificación de programadores (o en cualquier caso, clasificación de cualquier cosa) es que los sistemas de clasificación están basados en una serie de criterios. No hay un estándar real para clasificar programadores. Los sitios donde se clasifica y puntúa programadores en base a un número de problemas resueltos o la dificultad de los problemas resueltos solo han determinado que hay una serie de programadores que realmente disfrutan resolviendo esa clase de problemas. También han reunido a un grupo de programadores que dedicarán tiempo y esfuerzo a resolver estos problemas y serán competitivos mientras los resuelven. Nos dice poco sobre las habilidades del programador fuera de esa materia.

También hay otras medidas para clasificar a los programadores. Una medida clásica es revisar cuántas líneas de código usó un programador para resolver un problema (esto a veces se denomina “código de golf”, donde cuanto menor sea el número de líneas de código, mejor será la solución). Podemos argumentar acerca de cómo de “limpia” es la solución (limpia es otro término algo nebuloso y poco concreto). Podemos determinar la “notación Big O”, una notación utilizada para describir el rendimiento o la complejidad de los algoritmos que utilizó un programador en su código. Podemos hacer una prueba de *estrés* o de esfuerzo del código para determinar lo bien que se adapta el código a diversas circunstancias.

Podemos contar la cantidad de ciclos que requiere un código en particular para ejecutarse y compararlo con un código similar. Muy poco de esto nos dice algo sobre un programador en particular. Lo que sí nos dice es que el programador tiene experiencia que lo lleva a esa solución en particular. Nos dice que el programador ha visto este tipo de problemas antes y se preocupó lo suficiente por el problema como para pensar lo suficiente sobre cómo hacer una mejor solución. Aprendemos que el programador dedicó tiempo y energía a practicar este tipo de problemas. Lo que no nos muestra, es una medida general de las habilidades o capacidades del programador. Es similar al cuento apócrifo de un profesor brillante. Este profesor era un genio absoluto en su campo y era una de las personas a las que acudir para obtener respuestas sobre su tema, pero a pesar de su brillantez, no sabía cómo cambiar la rueda de un automóvil. ¿Significa eso que el profesor no era tan brillante como la gente decía que era? Difícilmente. Significa que el profesor pasó más tiempo pensando en su profesión que pensando en cambiar ruedas de automóvil. Lo mismo es cierto también para los programadores. Si un programador pasa la mayor parte de su tiempo resolviendo un conjunto particular de problemas, eventualmente se volverá experto en ese tipo de problemas. Pero si ese programador se atasca con un tipo diferente de problemas, eso no descarta sus habilidades generales, simplemente señala las áreas en las que podrían querer trabajar.

2.5 Midiendo el rendimiento del programador

También hay una tendencia a medir la productividad del programador mediante cuántas contribuciones puede hacer el programador a un proyecto. Bajo ciertos sistemas de control de versiones, estos se denominan *commits*. Enumeran un conjunto de cambios que el programador desea realizar en el código. En una era en la que existen sitios para escribir código de manera colaborativa y social como Github o Gitlab, podemos revisar fácilmente los *commits* que están realizando otros programadores. Dado que podemos medir la cantidad de *commits*, podemos usar esta medida para sentir que no estamos generando la misma cantidad y frecuencia de *commits* que otros programadores. Y a diferencia de las medidas de antaño (líneas de código

en particular, que mide cuántas líneas de código añade un programador a un programa), podemos revisar la calidad de sus *commits* con un proyecto. Puede ser desalentador ver una gran cantidad de trabajo de calidad realizado por nuestros compañeros. También puede ser fuente de frustración y sentimientos de insuficiencia. Nos preguntamos "¿Por qué no puedo ser tan productivo o contribuir como esta otra persona?"

Es incluso todavía más frustrante cuando otras personas utilizan estas medidas para judgar la productividad y las contribuciones con código. Podemos encontrarnos siendo criticados por nuestros resultados (o la falta de estos).

Los *commits* y las líneas de código son las medidas más visibles de la productividad al crear código, pero no muestran mucho sobre la práctica real de la programación. No podemos medir la cantidad de tiempo que dedicamos a pensar en el problema con solo mirar un *commit*. No vemos las montañas de material de referencia que usó el programador para encontrar una solución y ciertamente no sabemos si este *commit* es el resultado de una tarde de trabajo o de muchos días de trabajo (a menos que se hagan *commits* con más frecuencia). Incluso podríamos descubrir que esta persona está actuando como el punto central de una organización y está integrando el trabajo de varias personas en sus *commits*.

Medirnos a nosotros mismos basándonos en la cantidad de la producción de otras personas es fácil y muy seductor pero no es útil para hacerse una idea de cómo mejorar nosotros mismos en nuestra tarea de crear código (a no ser que sea "generando más *commits*"). Esa forma de pensar puede llevarnos a creer que no estamos dedicando el tiempo suficiente a la "programación real" y provocar exceso de trabajo, estrés y agotamiento.

2.6 Compañeros de viaje

Hay momentos en los que es útil compararnos con otros programadores. A veces podemos aprender sobre nuevas tecnologías o nuevas metodologías mirando el trabajo de otras personas, pero es fácil caer en la trampa de pensar que porque no estamos al nivel de otros programadores somos de

alguna manera inferiores a ellos. En lugar de ver a otros programadores como competencia, deberíamos verlos como compañeros. Todos estamos en esta profesión trabajando para mejorar nuestros respectivos proyectos. Con el software libre y de código abierto tenemos una oportunidad única de ver cómo otras personas hacen su trabajo en público. Podemos aprender del código de otras personas de la misma manera que los científicos pueden mirar los artículos de otros científicos para ver qué funcionó (y pueden mejorar la validez del artículo con una réplica y repetición). Ningún programador está completamente aislado del trabajo de los demás. (Es raro que un programador haya creado el código de todo su entorno de programación desde cero sin el trabajo de otras personas). Todas las personas aprendemos unas de otras, pero en lugar de intimidarnos por el trabajo de otras, podemos desarmarlo y aprender de él. Si tenemos suerte, podemos aprovechar la oportunidad para preguntarles cómo funciona el código y por qué eligieron escribir el código de esa manera.

Es valioso hacer preguntas a nuestros compañeros programadores. Tendemos a pasar por alto hacer preguntas por miedo a que vamos a hacer algo obvio o hacer una pregunta que nos haga sentir incómodos por haberla realizado. Hacer preguntas es muy útil cuando no entendemos qué está pasando con una idea o una pieza de código en particular. Hay programadores a los que no les importa responder preguntas, y espero que los encuentres. De acuerdo, hay algunos programadores que están muy ocupados y es posible que no tengan el tiempo o la predisposición para responder preguntas, pero si realmente estamos atascados y hemos agotado todas las demás vías, tal vez podamos hacerles preguntas que no requieran mucho de su tiempo y esfuerzo. Incluso pueden estar agradecidos por la pregunta porque les da una idea de una perspectiva que de otro modo no tendrían. Cuando hacemos preguntas iniciamos un intercambio de ideas en ambas direcciones.

Hacer preguntas es todo un arte y puede ser frustrante cuando las personas no responden nuestras preguntas o nos responden con otras preguntas o sugerencias que no son útiles. Esto se manifiesta en intercambios donde la persona A pregunta: “Me gustaría saber cómo hacer X” y las personas B y C responden “Yo en tu lugar haría Y”. Es frustrante cuando la gente no responde a nuestras preguntas de manera directa. También es fácil verse

envuelto en intercambios con la gente sobre los méritos de hacer Y, donde Y fue sugerido por otra persona que no tiene nada que ver con la pregunta original sobre X. Pero si volvemos a enmarcar la experiencia como “esta persona está tratando de ayudarme, tal vez haya algo en esta recomendación que pueda ser útil”, entonces podemos tener una mejor conversación. Quizás lo que estamos preguntando no es la mejor manera de hacer algo y hacer una pausa para escuchar puede ayudarnos a comprender mejor por qué hicieron su sugerencia.

El sacar nuestros egos de la pregunta nos permite estar más abiertos a las respuestas que recibimos. Cuando las personas no entienden nuestra pregunta, es fácil tomarla como algo personal y protestar pensando “no me entienden” o “no me escuchan”. Al abstraernos de la pregunta nos permite tomar la respuesta proporcionada “tal cual” y nos da la capacidad de cambiar la pregunta según sea necesario para obtener mejores respuestas.

Por supuesto, hay personas que no responderán pensando en lo mejor para ti y solo están interesadas en imponerte su propia visión del mundo. En lugar de responder a tu pregunta, cuestionan por qué estás haciendo eso y en su lugar sugieren que deberías usar su metodología. Puede requerir mucha energía enfrentarse con estas personas y decirles “no, realmente tenía la intención de aprender más sobre X”. Me gustaría tener buenas respuestas sobre cómo manejar a este tipo de personas. Muchas de estas personas, sienten que cualquier cosa que estén haciendo es el único camino correcto y aquellas personas que se desvían de su camino elegido son anatema para su mundo. Mi mejor sugerencia es agradecerles su tiempo y pedir ayuda a alguien más. Tal vez puedan ser útiles en el futuro cuando tenga preguntas sobre lo que sea que sea parte de su agenda, pero por ahora conviene ser lo más amable posible y desearles lo mejor en su viaje de programación. Los espacios tecnológicos tienen mucha gente que ha estado trabajando con computadoras durante mucho tiempo y se ha formado opiniones sólidas sobre sus herramientas y tecnologías. Mi esperanza es que puedas encontrar a las personas que también son amables y están dispuestas a compartir lo que saben y no acosarte con sus creencias arraigadas. Con el tiempo, también formarás tus propias creencias sobre lo que funciona y lo que no funciona y transmitirás ese conocimiento a los demás. Reconocer a las

personas que están ahí para ayudar a educar y a las que están ahí para hacer proselitismo es parte de nuestro proceso de crecimiento.

Si vemos a otros programadores como compañeros de viaje en esta travesía, como colegas en nuestra práctica a la hora de crear código, entonces nos daremos cuenta que estamos en esto todos juntos. Incluso alguien con muchos más años de experiencia que nosotros es tu colega. Tienes conocimientos y experiencia que ellos no tendrán y ellos tienen experiencias y conocimientos que tu no tienes. Si nos despojamos de las barreras del estatus que percibimos y de la meritocracia de esas personas podremos entendernos mejor y aprender unos de otros.

El viaje para llegar a ser un programador mejor es largo y duro. Necesitamos los mejores compañeros que podamos encontrar para ayudarnos en este viaje. Necesitamos algo más que únicamente compañeros que tengan muy buenos conocimientos técnicos, también necesitamos compañeros con los que podamos hablar cuando la jornada termina. Necesitamos compañeros con los que podamos sentarnos alrededor de la fogata proverbial e imaginaria donde juntos podamos reír y compadecernos de nuestras luchas.

3 Los errores a lo largo del camino

3.1 ¡Vaya!

Tiene que pasar: algo que pensabas que era una buena idea no funcionó de la manera que lo habías planeado, y ahora te das cuenta de que has cometido un terrible error. A veces es algo que se podría haber evitado fácilmente (por ejemplo: realizar un *commit* de un código destinado a la depuración). A veces es una cascada de errores, cada uno de los cuales se basa en las consecuencias del error anterior. Está el error de ignorar los efectos secundarios de un módulo cuando se usa de una manera que no estaba prevista, o darse cuenta de que has diseñado un módulo pequeño y estrechamente acoplado y después descubrir que tu módulo será parte de una pieza de software más grande y tu código no está diseñado para hacer una transición sin problemas. ¡Vaya!

Sin embargo, los errores que realmente me asustan son los que no esperaba, aquellos en los que las consecuencias no deseadas corren desenfrenadas por todo el sistema como una reacción en cadena. Esos errores son los que me quitan el sueño por la noche.

Los programadores cometen errores. La naturaleza de nuestros trabajos requiere que seamos conscientes de lo que sucede en múltiples secciones de código. Perdemos la noción del estado de nuestro programa y del código que ya hemos realizado. Intentamos salpicar nuestro código con comentarios y recordatorios de lo que sucede en una sección del código, pero los comentarios se vuelven obsoletos y aumentan nuestra distracción. Nos apresuramos y confiamos que el músculo de nuestra memoria tome el relevo. Nos negamos áreas en las que podemos probar el código adecuadamente porque sentimos que debemos apresurarnos para hacer las cosas rápidamente.

Entramos en pánico, y cuando entramos en pánico cometemos errores.

3.2 Evitar los errores

Vamos a hablar con claridad: no hay manera de evitar o eliminar los errores. El software es demasiado complejo para estar completamente libre de errores. Sin embargo, lo que podemos hacer es crear lugares en los que podamos detectar tantos errores del código como sea posible antes de mostrarlo a los demás. Podemos comprender mejor nuestro código y lo que está haciendo cuando tenemos la capacidad de depurar y probar nuestro código en un entorno seguro. Podemos ver cómo se comportará bajo ciertas circunstancias. La creación de un modelo similar al del sistema de destino nos permite probar nuestro código con versiones más pequeñas de la realidad del sistema del destino y ver cómo se comporta en esas condiciones.

Ponemos mucho énfasis en evitar errores, tanto en la programación como en la cultura de la programación. Hay historias de terror de cómo pequeños errores en un programa causaron grandes dolores de cabeza a las personas involucradas. La moraleja de estas historias es que los errores simples pueden ser costosos y debemos ser doblemente cuidadosos para evitar errores. Estas anécdotas se cuentan con la esperanza de asustar a los desarrolladores para que sean más cautelosos, pero pueden tener el efecto contrario. Pueden hacer que los programadores se vuelvan paranoicos acerca de poder cometer cualquier error, y cuando operamos en un modo basado en el miedo, comenzamos a entrar en pánico. Decirle a los programadores que no cometan errores es similar a decirle a alguien que no tenga miedo: tienen más miedo de tener miedo.

La mejor (y quizás la única) forma en que aprendemos, es cometiendo errores. Aprender cometiendo errores es una forma efectiva de permitirnos ser curiosos y ver qué causó que el programa fallara. Cuando nos privamos de la libertad de cometer errores, nos privamos de las oportunidades de aprendizaje al cometer esos errores. Eso no significa que tengamos que cometer todos los errores que otros desarrolladores han cometido antes que nosotros (eso serían montones de errores). Tampoco significa que debamos introducir el caos en nuestro proceso de desarrollo para aprender mejor. Lo que significa es que debemos cometer nuestros propios errores a nuestra

manera, para seguir aprendiendo y descubrir dónde existen lagunas en nuestra comprensión.

3.3 Hacer un modelo

Necesitamos entornos donde los programadores puedan aprender de una manera segura de sus propios errores. Necesitamos espacios donde los programadores puedan sentirse bien y seguros a la hora de probar cosas nuevas. Necesitamos lugares donde los desarrolladores puedan probar sus ideas y que esos cambios no se extiendan a otros sistemas no relacionados. Esta es la mejor forma, con la que los desarrolladores pueden aprender y ser valientes en su proceso de aprendizaje.

Estos entornos deben replicar los sistemas de destino y deben estar lo más cerca posible de esos sistemas de destino. Eso no significa que haya que hacer copias exactas de entornos de producción costosos, pero sí se necesita crear modelos de entornos de producción que prueben la mayoría de las piezas con las que tu código entrará en contacto. Tener modelos o réplicas que reflejen los sistemas de producción significa que cuando muevas tu código a producción, introducirá menos cambios con consecuencias no deseadas. Tus cambios ya habrán existido en un entorno de producción. Puedes estar tranquilo sabiendo que los cambios que provoquen en estos modelos serán los mismos cambios que aparecerán en el sistema de destino.

En la situación más ideal, necesitarás tener un entorno como este, en una máquina que controles. Esto significa que no estás compitiendo con otros programadores de tu organización que también están siendo valientes con sus cambios. También querrás asegurarte de que tu entorno se mantenga actualizado con sus cambios (y cualquier cambio de producción) para que tu modelo de desarrollo coincida con lo que está en el sistema de destino y lo que estará en el sistema de destino. Un buen modelo es aquel que se mantiene actualizado con lo que se está modelando. Es lo mismo que un mapa de una ciudad: es mejor cuando coincide con el área de su modelo y se mantiene actualizado con los cambios que ocurren en esa ciudad. Un buen mapa de la ciudad podría informarte sobre las obras recientes que se están realizando en tu ruta. Un mapa inútil es aquel que ni siquiera muestra

tu ruta porque no se había construido esa ruta cuando se creó el mapa. Si nuestro modelo de producción se está quedando atrás constantemente con respecto a lo que está en producción, dedicaremos más tiempo a rectificar los cambios que estamos haciendo con los cambios entre nuestro modelo y el que está en producción.

Esto también significa que deberíamos tener un entorno que se pueda reconstruir rápidamente y replicar según sea necesario. Tener un modelo que se convierte en su propia realidad separada se convierte en un sistema más para mantener. Este modelo debe ser algo que se pueda eliminar y reconstruir a voluntad para eliminar cualquier experimento anterior. Es mejor pensar en él como una copia efímera de tu entorno de destino que tiene un uso limitado y puede desecharse cuando ya no sea necesario. Este entorno de pruebas debe ser rápido a la hora de volverlo a replicar para que haya poca fricción a la hora de crear nuevos entornos para probar. Eso quizás puede significar crear secuencias de comandos para el proceso de construcción de estos entornos. La forma en que decidas hacer esto depende de ti, pero ten en cuenta que quieres algo que sea lo más simple posible y que requiera la menor cantidad posible de nuestra atención para replicarlo.

De nuevo, no tiene que ser perfecto, sólo es un modelo, pero necesita ser lo bastante similar al original para que tu código se comporte de una manera similar entre el modelo creado y el entorno real.

3.4 Máquinas del tiempo

Hay un buen número de personas que te hablarán sobre los beneficios de un sistema de control de revisiones (y muchas de esas personas te mostrarán los pasos exactos para configurar un sistema de control de revisiones). Los sistemas de control de revisiones como `git`, `svn`, o `cvs` y similares han ayudado a los programadores a coordinar lanzamientos de publicaciones y mantener un registro de qué trabajos se han añadido a sus proyectos. Tener un buen sistema de control de revisiones te permite crear áreas donde puedas probar código nuevo sin tener que añadir estas pruebas a código ya en producción. Un buen control de revisiones te permite crear un espacio (o también llamados ramas o `branch` por su nombre en inglés cuando estamos

hablando de `git`) basado en un código ya existente que puedes utilizar para experimentar y desarrollar. También te permite realizar *commits* en ese espacio y divagar tanto como necesites para poder explorar a fondo los cambios que estás realizando. Sin embargo, lo que es más importante es que un buen sistema de control de revisiones también te permitirá abandonar ese espacio si lo necesitas, no estás forzado a añadir esos cambios a tu código en producción. Esto te permite ver si algo podría funcionar o abandonar esos cambios si no lo hace. Un buen control de revisiones brinda a los programadores la capacidad de ramificarse desde cualquier punto en el tiempo y explorar lo que sucedió en el código base. En cierto sentido, son máquinas del tiempo y universos infinitos, lo que te permite jugar con distintos escenarios "¿y si?" con tu código y avanzar y retroceder en el tiempo en el código. Esto es vital para tu aprendizaje porque puedes sentirte seguro probando y probando cosas y rebobinando esos cambios (o eliminándolos por completo) sin afectar el trabajo de otras personas.

Aprender cómo funciona tu sistema de control de revisiones te dará libertad para cometer errores. Muchos de estos sistemas pueden parecer complejos al principio, pero con la práctica continua y paciencia, comprenderás lo que hace el sistema de control de revisiones y cuáles son sus posibilidades. Podrás juzgar cuántos riesgos puedes tomar con tu código y tener más confianza con los riesgos que tomas.

El control de revisiones también puede desempeñar un papel importante al poder ver el desarrollo del código de otras personas. Puedes disponer de una ventana a su proceso de desarrollo y ver cómo se desarrollan ciertas características a medida que se añaden. Esto puede ayudarte a aprender sobre una base de código desconocida y mostrarte la dirección que tomaron para hacer el código de la manera que es. Puede brindarte una ventana a la historia de un proyecto y lo que se hizo para que sucediera. El control de revisiones puede ser una máquina del tiempo en la historia de un proyecto y puede ayudarte a comprender que la programación es un proceso. No todos los proyectos vienen completamente ya formados de la mente de los programadores.

3.5 Aprender de los errores

A veces fallamos. A veces, el código que escribimos no está a la altura de las realidades del sistema en el que se implementa. Enviamos código que hace algo inesperado y como resultado, los sistemas se rompen. Podemos perder la noción de dónde estamos en nuestro código y hacer cambios que entren en conflicto con otros cambios, lo que luego hace que dediquemos tiempo a deshacer esos cambios. Todos estos casos causan malestar, ya sea para nosotros, las personas a las que ayudamos o las personas con las que trabajamos.

No voy a mentir: el fracaso apesta. Nos hace sentir que somos menos personas porque fallamos. Nos sentimos incómodos y nos preguntamos qué pensarán los demás de nosotros. ¿Tendrán los demás una mala opinión sobre nosotros? ¿Hemos dañado nuestra relación con aquellas personas que usan lo que sea que hayamos programado? ¿Hemos defraudado a nuestro equipo? Todas estas preguntas surgen de dos deseos: el deseo de hacer lo mejor posible y el deseo de no hacer daño a los demás. Queremos que los demás piensen bien de nosotros y de nuestras habilidades. El fracaso va en contra de esos deseos y amplifica cualquier sentimiento de insuficiencia que podamos tener. Esos sentimientos pueden incluir preguntarse si deberíamos programar o si nuestros talentos deberían usarse en otras áreas. Nos preguntamos si deberíamos rendirnos.

No solemos pensar en el fracaso como parte del proceso de aprendizaje. El fracaso a menudo se ve como el punto final del viaje. En la escuela, una nota baja se considera una castigo. No lo vemos como: “necesito practicar esto un poco más”, en cambio, sentimos que nos hemos causado vergüenza e incomodidad a nosotros mismos y a nuestros seres queridos. Nos perjudicamos gravemente a nosotros mismos, si no nos damos cuenta de que el fracaso es una parte natural del proceso de aprendizaje y que está bien fracasar. No todo lo que hagamos será perfecto. Los errores se infiltrarán en el mejor código que escribamos. Nos equivocaremos y desplegaremos el código en el sistema equivocado. Nuestros errores causarán malestar a los demás. Aceptar esto nos da la libertad de darnos cuenta que, a pesar de nuestros mejores esfuerzos, no seremos perfectos. En lugar de ver el fracaso como una limitación, podemos usarlo como parte de nuestro proceso de crecimiento.

Cuando nos damos cuenta de que vamos a cometer errores, podemos cambiar nuestro enfoque sobre cómo y dónde los cometemos. Antes mencioné sobre la creación de modelos de nuestros entornos. ¿Qué mejor manera de permitirnos cometer errores que en un entorno donde esos errores se pueden contener o revertir? La creación de modelos nos permite practicar y probar nuestras suposiciones en entornos que nadie más tiene que ver. Es similar a un lugar de ensayo para músicos, donde pueden repasar su material sin necesidad de tocarlo bien a la primera. Pueden resolver las partes problemáticas y cometer errores hasta que tengan confianza en su ejecución.

Los errores sirven para aprender qué funciona y qué no funciona. Son una parte integral de nuestro proceso de aprendizaje. Tenemos tendencia a recordar las lecciones de aquello que no funcionó bien, mejor que aquellas que sí funcionaron. Los errores nos ayudan a reforzar aquellas áreas donde nos faltan conocimientos y nos ayudan a comprender las brechas que aún tenemos que cerrar.

Los errores también actúan como un recordatorio para hacer una pausa por un momento y no dejarse llevar por la urgencia de las cosas. Mis propios errores tienden a surgir cuando me apresuro a cumplir con una fecha límite (ya sea real o auto impuesta). Mis peores errores suceden cuando estoy cansado y apurado, cuando prácticamente estoy golpeando el teclado tratando de hacer que algo (¡cualquier cosa!) funcione. Cuando me permito hacer una pausa por un momento, reflexionar sobre lo que estoy tratando de hacer y sentir la incertidumbre en el momento, puedo tomar medidas para recalibrar y reenfocarme en el momento. Me doy la libertad de corregir el rumbo y comprender que no estoy dando lo mejor de mí y necesito hacer algo diferente. Puede ser algo pequeño como darle un poco de descanso a mi cerebro o algo grande como revisar las suposiciones que hice sobre lo que estoy haciendo. Hacer la pausa me permite determinar si quiero continuar haciendo lo que estoy haciendo y entender si ese es el mejor camino.

3.6 Llevar un registro de nuestros errores

Es interesante no cometer los mismos errores dos veces, pero incluso si repetimos el mismo error aún puede ser útil. Saber que hemos repetido el mismo fallo es útil porque nos da un patrón que podemos entender. Esos patrones nos muestran que hacer esto en particular conduce a un resultado erróneo que se vuelve a repetir. Luego podemos determinar qué causó el error y planificar cómo mitigarlo. Esto es parte del proceso de aprendizaje, siempre y cuando no caigamos en una espiral de auto-recriminación cuando nos demos cuenta de que hemos vuelto a cometer el mismo error.

Un truco que yo mismo debería usar con más frecuencia es escribir un registro en un diario. Llevar un diario de lo que ocurrió y cómo lo solucionamos es una forma de explicarle a otra persona (a menudo a nosotros mismos) lo que sucedió. Explicar lo sucedido nos permite convertirnos en maestros para nosotros mismos y para los demás. Refuerza nuestro proceso de aprendizaje. Escribir lo que sucedió de una manera que otros puedan entender, nos permite organizar los pensamientos en nuestra cabeza de una manera clara y comprensible. Cuando articulamos nuestros propios pensamientos sobre lo que sucedió y los transcribimos, comenzamos a comprender nuestros propios pensamientos y podemos liberarnos de otras ideas sobre cómo solucionar este y otros problemas. Nos damos la pausa que necesitamos para comprender completamente lo que sucedió y cuál es la mejor manera de avanzar. Nos convertimos en nuestra propia caja de resonancia de ideas sobre la mejor manera de proceder.

No se trata de mantener un registro para la posteridad para que podamos mirar hacia atrás en una lista de fallos y castigarnos por el pasado (si eres como yo, eso sucede automáticamente). Es una manera de enseñarnos a nosotros mismos y maximizar el proceso de aprendizaje. Se trata de darnos la libertad de ser el instructor de nuestro yo futuro para que podamos ser más conscientes cuando un error está a punto de ocurrir y comprender cómo corregirlo. Esto nos permite concentrarnos ahora mismo el tiempo suficiente para comprender qué sucedió, qué hicimos para corregirlo y cuál es la mejor manera de proceder a partir de aquí. También nos ayuda a ubicar dónde están nuestras lagunas de aprendizaje y las “próximas acciones” que debemos tomar para llenar esas brechas.

Hablaremos más sobre el proceso de llevar un registro en capítulos posteriores pero recomiendo encarecidamente el hábito de llevar un diario, aunque únicamente sea por la razón que le brinda un aprendiz dispuesto a enseñar, incluso si ese aprendiz eres tu mismo.

4 Las posadas en las que nos hospedamos

4.1 Compañeros de viaje

Siempre que pensamos en programadores tendemos a pensar en una persona sentada frente a un ordenador escribiendo código, con el brillo del monitor reflejándose en su cara. Normalmente el programador está solo (aunque hay algunas metodologías en las que participa más de un programador a la vez, “programación en pareja” o “pair-programming” por ejemplo). Durante esas sesiones de escribir código, no hay mucho contacto con otros programadores y puedes sentirte aislado al estar en compañía únicamente de ti mismo. De acuerdo, esta puede ser una sensación agradable (hay momentos en los que realmente disfruto de la soledad junto al ordenador, completamente ocupado y centrado), pero hay otros momentos en los que necesitamos sentir que no estamos solos. Esto es especialmente cierto cuando estamos aprendiendo y adentrándonos en un territorio incómodo. Encontrar a otras personas en situaciones similares nos puede ayudar en nuestro proceso de aprendizaje. Otras personas nos pueden ayudar respondiendo a nuestras preguntas y revisando nuestro progreso. Encontrar una buena comunidad que nos apoye en nuestro aprendizaje es esencial en nuestro viaje en la programación. Cuando tenemos una gran comunidad, tenemos un lugar donde poder aprender y ayudar a que otras personas aprendan. Podemos crecer en la comunidad y encontrar apoyo.

Una buena comunidad es aquella que nos fortalece a nosotros y a quienes nos rodean. Nos nutre y nos proporciona refugio. Es un lugar seguro donde no tenemos que mantener la guardia alta de los ataques a nosotros mismos y a los demás. En ella las personas se rinden cuentas entre sí. Podemos confiar en los miembros de la comunidad y sentir que la confianza es recíproca. Las buenas comunidades existen sin competencia ni ego, donde los miembros pueden expresarse abiertamente y aceptar a los demás tal como son.

4.2 Encontrar una buena comunidad

Hay un montón de buenas comunidades ahí fuera que están dispuestas a ayudarte a ser un mejor programador, pero ¿cómo encontrarlas?

Esa es una pregunta complicada.

La mayoría de lenguajes de programación tienen algún tipo de comunidad que se ha formado entorno a ellos. Algunas usan las listas de correo u otros canales de comunicación a los que te puedes unir y en los que participar. Desafortunadamente, los lenguajes de programación más populares tienen espacios que son complicados de seguir, especialmente cuando estás tratando de aprender. Lo sé porque me uní al canal principal de comunicación de un lenguaje de programación muy popular y me sentí abrumado por las múltiples conversaciones que se llevaban a cabo a la vez. Las listas de correo diseñadas para ayudar a los principiantes pueden tener una gran cantidad de tráfico de correos y ese tráfico puede ser abrumador si estás tratando de entender los aspectos básicos del lenguaje de programación mientras tratas de estar al día con la avalancha de correos que llegan a tu bandeja de entrada. Leer los correos antiguos en los archivos de las listas de correo o en un chat puede ayudarte a determinar si estás preparado para ese nivel de tráfico y si las conversaciones en la lista son el tipo de conversaciones que te gustan. Recuerda: esto es para ayudarte en el viaje. Lanzarte a una habitación abarrotada, en la que te veas inundado por una gran cantidad de conversaciones y ruido solo hará que te sientas más aislado y sentir que no eres bienvenido.

Algunos lenguajes de programación tienen grupos de usuarios locales. Estos al principio pueden parecen intimidantes, especialmente si el grupo ya tiene un largo recorrido. Lo sé, porque me sentí intimidado antes de asistir a mi primer grupo local de usuarios, por miedo a lo que podría encontrarme allí. Lo que encontré fue un grupo de personas que estaban interesadas en los temas en los que también yo estaba interesado. He hecho amistades duraderas en esos grupos locales de usuarios y te animo a comprobar si también a ti te funciona.

Si no sabes cómo encontrar el grupo adecuado (quizás te encuentres en un área en la que sientes que eres la única persona que comparte tus intereses), podrías considerar iniciar tu propio grupo o formar uno nuevo a partir de un grupo ya existente. Mi amigo Rick y yo abrimos un grupo local llamado *Coffee House Coders* donde los programadores se reúnen una vez a la semana durante unas horas para sentarse y programar. Todo lo que hicimos fue publicar las horas y los lugares en los que nos íbamos a reunir y les dijimos a las personas que simplemente se presentaran con su portátil para escribir código. Por el camino hemos conocido a algunas personas increíbles y hemos mantenido al grupo en marcha durante muchos años. Empezar un grupo es un acto de valentía. Ha habido muchas noches en las que me he sentado solo en una cafetería esperando a ver si aparecían otras personas. Está bien. La gente está ocupada y los intereses se desvanecen y cambian con el tiempo. Lo importante es crear el espacio para que nosotros y los demás nos sintamos bienvenidos. Para nosotros, eso significaba encontrar una cafetería local que estuviera abierta hasta altas horas de la noche y tuviera un amplio espacio para colocar nuestros portátiles. También ayuda encontrar un lugar en el que haya suficientes tomas de corriente para conectarse a la energía eléctrica, para que la gente pueda cargar las baterías de sus equipos si es necesario.

Hay muchas maneras de ser creativo a la hora de comenzar una comunidad. La llegada de las herramientas en la red permite que puedas construir comunidades con personas de todo el mundo. El acercar esas personas y reunirlas para hablar y discutir ideas y ofrecer ayuda es genial cuando ocurre. A veces, puede ser tan simple como crear una sala de chat sobre intereses comunes. Explora lo que existe ahí fuera y si lo que hay no satisface tus necesidades no dudes en crear una tu mismo.

4.3 La dificultad de encontrar una buena comunidad

Reconozco que cualquiera no puede unirse a una comunidad o crear una propia. Los espacios existentes en la red tienen la reputación de no ser lugares acogedores para las personas que llegan y los encuentros en persona

pueden agotar tus recursos mentales. Me llevó tiempo encontrar el valor necesario para asistir a mi primer encuentro en persona y tuve una mala experiencia con alguien con el que trabajaba y que pensé que estaría en aquel encuentro. (No estoy seguro si esa persona alguna vez asistió a alguno de aquellos encuentros). Pero estoy muy agradecido de haber asistido a aquellos primeros encuentros. Asistir a esos eventos me trajo amistades, oportunidades y otros "compañeros de viaje" para mi propio viaje. Hizo que cambiara a uno de mis lenguajes de programación favoritos (Python) y me llevó a varios trabajos. También me ayudó a sentir que no estaba solo con mis intereses y me presentó a otras personas en las que podía confiar. Me dio un sentimiento de pertenencia.

Superar el obstáculo inicial es difícil. Nuestro miedo al rechazo y nuestro miedo a hacernos vulnerables a los extraños puede desgastarnos. Superar ese miedo requiere mucha energía mental y puede quitarnos el deseo de ser parte de otra comunidad más. No puedo decir que será fácil, pero puedo señalar algunos de los beneficios que tuvo en mi vida. Espero que tu puedas encontrar también esos beneficios.

Una alternativa a las comunidades en persona son las comunidades en la red. Las comunidades en la red pueden ser una excelente manera de encontrar a otras personas. Reúnen a personas de diferentes países y las reúnen en un área común. Una parte de las razones que hicieron que cambiara a encuentros en persona fue por las buenas experiencias que tuve con estas personas en salas IRC (Internet Relay Chat). Disfruté de la compañía de estas personas a través de interacciones por la red y me sentí cómodo cuando las encontré en persona.

Los bajos requisitos de entrada para muchas comunidades en línea, puede permitirnos ver de qué temas se trata en la comunidad. ¿Cuáles son sus prioridades? ¿Son amables con las personas que están comenzando? ¿Tienen un patrón de conducta de ayuda a personas como nosotros o tienden a ofenderlas? ¿Tienen miembros que nutren a sus compañeros o se están interrumpiendo unos a otros?

Desconozco si hay una estrategia óptima para determinar si una comunidad es útil o dañina. Requiere esfuerzo seguir una comunidad y formarse una idea de quiénes son. Desgasta emocionalmente el ponernos en situaciones

donde somos vulnerables, para comprobar si otras personas son respetuosas con nosotros. Las comunidades están formadas por personas y las personas somos criaturas inconstantes e irracionales. Lo que puede ser una gran comunidad para una persona puede ser un entorno tóxico para otra. Aunque no tengo una estrategia, tengo algunas ideas de los elementos clave que forman una comunidad.

4.4 Cosas que buscar en una buena comunidad

Hay una serie de características que yo buscaría en una comunidad. Aunque esta no es una lista definitiva de todas las cosas que hacen una buena comunidad, ofrece un listado de las pautas que creo que son importantes:

- **Código de conducta**: Las buenas comunidades tienen pautas para las cosas que la comunidad aceptará, tolerará o impedirá. Debe ser visible para todos los miembros de la comunidad, y cada miembro de la comunidad debe ser responsable de esas pautas. También se deben hacer cumplir. Si observas situaciones en las que el código de conducta se aplica de manera colectiva contra ciertos miembros, debes tener cuidado de permanecer dentro de esa comunidad.
- **Moderadores**: Se necesita que haya una persona (o un grupo de personas) en la comunidad que pueda calmar las situaciones de conflicto y aplicar castigos significativos cuando las personas se extralimiten. Los moderadores deberían ser imparciales y consecuentes con sus decisiones. Deberían demostrar que también siguen el mismo código de conducta con el ejemplo de sus acciones en la comunidad. Un buen moderador debería estar siempre disponible, pero no ser autoritario. Deberías sentirte cómodo con la presencia de un moderador y sentirte libre de preguntar a los moderadores si tienes alguna duda sobre la comunidad.
- **Espacios para preguntas y pautas para realizar buenas preguntas**: Debería existir un lugar donde las personas puedan preguntar cuestiones relacionadas con la comunidad. Las personas deberían sentirse seguras al preguntar sobre el tema y la comunidad necesita ser

clara sobre lo que considera una pregunta apropiada e idónea. ¿Es el sitio adecuado para preguntas de recién llegados? Si no lo es, ¿se podría crear un sitio así? ¿Qué clases de preguntas estaría la comunidad feliz de responder y qué tipos de preguntas molestarían a la comunidad? Esto necesita estar claramente definido para que las personas recién llegadas puedan hacerse una idea de qué será bienvenido por la comunidad y qué no tolerará.

- **Disfrutar**: ¿Disfrutan las personas que forman la comunidad al discutir las cosas? ¿Cual es el tono de las conversaciones? ¿Interactúan las personas de una manera positiva con otras o recurren a los insultos y las faltas de respeto? ¿Se celebran las nuevas preguntas planteadas o son rechazadas e ignoradas? Si no se disfruta al ser parte de la comunidad, entonces se reducirá el número de personas que se queden en ella.
- **Compasión y empatía**: ¿Permite la comunidad que la gente cometa errores? Cuando algo sale mal, ¿la comunidad trata de ayudar? ¿La comunidad recuerda cómo era ser principiante y actúa con compasión, o espera que todos tengan más experiencia antes de participar?
- **Amabilidad**: Este es el factor más importante, ¿la comunidad se comporta de una manera amable con los demás o se divide en facciones y tratan de reducirse entre sí? ¿Ven a la gente nueva como amigos o como extraños que deben probarse a sí mismos? Esto se relaciona con la compasión y la empatía anteriores, pero tendemos a ver actos de bondad antes de ver la compasión y la empatía. La amabilidad se manifiesta cuando los miembros de la comunidad comprenden que las personas no sepan todo de inmediato y actúan con amabilidad en lugar de adoptar un enfoque severo. Les hacen saber a las personas que ellos también han tenido problemas y sugieren formas de trabajar juntos para suavizar las cosas para las próximas personas que puedan experimentar el mismo problema. Actúan de una manera que no antepone su ego y, en cambio, se comportan como si les hubieran dado un regalo que es mejor compartir con los demás.

Hablaremos más sobre la amabilidad en próximos capítulos.

Estos son solo unos ejemplos de lo que he encontrado en buenas comunidades. No dudes en añadir más a esta lista cuando tu propia

experiencia vaya creciendo y házmelo saber, así podré actualizar esta lista para futuros lectores.

5 Un día de viaje

5.1 Cabalgando hasta el amanecer

Como programadores siempre estamos tratando de encontrar nuevas formas de ser productivos. Hacemos ajustes en los editores de texto, ajustes en la compilación, creamos *scripts* y automatizamos tareas, la lista se extiende dependiendo de cómo los programadores quieran maximizar la productividad del tiempo que emplean en crear código. También pasamos tiempo haciendo ajustes al resto de nuestras vidas con la creencia de que deberíamos siempre estar haciendo algo relacionado con el código. Cualquier momento que no estamos creando código es un momento en el que nuestros proyectos se retrasan. Y retrasarse con nuestro código puede desencadenar en otros problemas: plazos incumplidos, que otras empresas publiquen su programa en el mercado antes que nosotros, u otros casos en los que perdemos una oportunidad. Estamos constantemente preocupados por no estar haciendo lo suficiente para tener éxito.

Hemos escuchado historias de desarrolladores que se han despertado frente a sus equipos por un extraño pitido provocado porque se han quedado dormidos encima del teclado y la repetición automática de caracteres ya no podía gestionar más texto por que sus caras estaban apoyadas en las teclas. ¿No es así como deberían trabajar los desarrolladores?

Hay una tendencia a creer que como trabajamos con máquinas que no descansan y siempre están preparadas para trabajar más, necesitamos adaptarnos nosotros mismos a esas máquinas. Sentimos la necesidad de estar siempre en marcha y preparados para dar a la máquina más trabajo. La ociosidad se considera una pérdida de tiempo. Tratamos de llegar a ser como la máquina: sin descanso y siempre preparados para más trabajo.

Hay un problema con ese sentimiento de tener que estar siempre en marcha. Cuando sentimos que siempre tenemos que estar en marcha, no nos permitimos a nosotros mismos tener un momento de descanso. No nos

permitimos periodos de ocio y descanso. Esto crea un patrón donde nos denegamos los momentos para sentarnos y centrarnos en lo que estamos haciendo. Nos forzamos a seguir en movimiento, a seguir programando sin importar el coste personal. Nuestros cerebros no tienen la habilidad de descansar, relajarse y cargarse de energía. Nuestras mentes están ocupadas y exhaustas para procesar lo que aprendimos y guardar ese conocimiento a largo plazo. Cuando estamos exhaustos empezamos a preocuparnos de que no estamos haciendo todo lo necesario. Esto no nos motiva, en vez de eso crea un círculo vicioso de miedo y pánico. Pasamos nuestro día preocupados porque no estamos haciendo lo necesario mientras que nuestras mentes gritan un "¡suficiente!" de extenuación. Este círculo vicioso de miedo y extenuación puede llevarnos a un vórtice de agotamiento, depresión y el deseo de dejar la programación para siempre.

Hay un delicado equilibrio que debemos lograr, entre nuestros deseos de estar conectados constantemente y nuestras necesidades de relajación y reflexión. Nuestro deseo de un desarrollo invencible e infatigable debe atemperarse con la realidad de que nuestros cuerpos y mentes tienen unos recursos finitos por día que deben asignarse adecuadamente. Piensa en esto, como en el suministro de energía para una máquina compleja (nuestro cuerpo y mente) en la que el fabricante (todavía) no te permite cambiar la batería cuando se agota. Ser consciente de qué procesos se están ejecutando, cuánta energía se está utilizando y cuánta energía queda es vital para garantizar que aún pueda funcionar más, hasta el final del día. Ese es el nivel de conciencia que debemos tener sobre nosotros mismos.

¿Cómo equilibramos esos sentimientos de querer estar conectados todo el tiempo mientras nos permitimos a nosotros mismos el relajarnos y reflexionar en lo que estamos haciendo? ¿Cómo prestamos atención a las necesidades de esta "máquina de programar"?

5.2 Luces fuera

Primero, debemos ser conscientes de que no podemos estar conectados todo el tiempo. Quizá sabes esto de manera intuitiva y pensamos "claro, por supuesto" pero saber esto no es lo mismo que ponerlo en práctica.

Necesitamos periodos donde no estemos programando ni pensando en programar. Deberíamos tener momentos donde poder apagar la parte del programador de nuestra personalidad. Estos periodos de no programación son vitales para nuestro propio bien estar y nos ofrecen oportunidades para explorar el ancho mundo y permitir a nuestras mentes descansar en los tiempos entre las sesiones de programación.

Esto puede ser complicado si sentimos que nos estamos retrasando en nuestro aprendizaje. ¿Cuando se supone que vamos a aprender todas esas cosas nuevas que están ocurriendo cada día? ¿Cuándo se supone que nos pondremos al día con toda esa deuda técnica que hemos ido acumulando a lo largo de los años? ¿Cuándo tendremos tiempo de conocer los entresijos de tecnologías que no forman parte de nuestro trabajo diario pero que nos siguen interesando?

Estos sentimientos que tenemos (que hay que hacer más y que necesitamos pasar cada momento en el que estamos despiertos haciendo cosas que no hagan que nos desfasemos) no son muy útiles cuando nos comparamos con otros programadores que parece que son súper productivos. Estos son los programadores que tienen una idea brillante por la mañana y un prototipo ya funcional por la tarde (mientras siguen gestionando normalmente su rutina de trabajo). Cuando nos comparamos a nosotros mismos con estos programadores nos preguntamos si se toman algún momento libre lejos de sus ordenadores.

Podemos reconocer que tenemos sentimientos de querer esforzarnos para seguir aprendiendo y haciendo cosas. Podemos notar nuestros sentimientos cuando pensamos “solo una línea de código más antes de acostarnos” o convencernos a nosotros mismos pensando: “puedo leer algunos artículos o páginas más o [inserte la forma favorita de consumir más información aquí]”. Podemos hacer una pausa y notar de dónde vienen estos sentimientos y pensamientos y entender por qué incluso nos esforzamos más allá del agotamiento.

Estos sentimientos generalmente provienen de una sensación de insuficiencia. Sentimos que no estamos a la altura de los ideales que tenemos, ya sea porque estos ideales son los que hemos creado o por que han sido creados de manera externa. Estos ideales provienen de analizar a

otros programadores (colegas o personas a las que admiramos) y medir nuestro progreso en comparación con su trabajo. También provienen de nuestras propias ideas míticas de lo que hace a un programador perfecto.

De lo que debemos darnos cuenta es que esas ideas de lo que hacen los programadores buenos y perfectos son fantasías. Son una combinación de lo que creemos que debería ser un programador bueno y perfecto. No existen en el mundo real. Es cierto que podemos ver programadores que parecen despertarse con un teclado conectado a sus manos, pasan todo el día creando código y se van a dormir con sueños de formular más código en sus cabezas. Pero debemos darnos cuenta de que solo estamos viendo un lado de sus vidas. No estamos viendo la imagen completa de quiénes son. Necesitamos enfocarnos en nuestros propios cuerpos y mentes y darnos cuenta de cuándo estamos cansados y necesitamos descansar. No podemos convertirnos en otras personas, tenemos que trabajar con quienes somos y lo que somos.

Nuestros cuerpos necesitan tiempos de descanso para poder ser más efectivos. Necesitamos momentos donde podamos apartarnos del teclado y permitirnos a nosotros mismos descansar y relajarnos. Nuestras mentes no están diseñadas para el trabajo constante, especialmente a los niveles que se le requiere a un programador. Cuanto antes nos demos cuenta de que debemos dar un paso atrás y tomar descansos a lo largo del día para recargarnos, más felices (y más productivos) seremos.

5.3 Hacer un descanso

Hacer un descanso es algo más que simplemente cambiar a otro programa de tu ordenador. Mi tendencia mientras tomo un descanso es comenzar a revisar mi correo electrónico o abrir alguno de mis diversos programas de chat para ver qué ha pasado desde la última vez que lo abrí (normalmente desde el último descanso que me tomé). Esto realmente no es tomarse un descanso ya que estoy tratando de realizar múltiples tareas sentado en mi mesa. Los verdaderos descansos implican levantarse de delante de la pantalla del ordenador. No es necesario que sea un descanso largo, tomarse un descanso puede ser simplemente salir de la habitación a otra área

diferente de donde estés trabajando. Necesitas moverte de delante de tu equipo para obtener un “cambio de contexto”, donde tu mente puede sentir que no está en el mismo lugar donde estaba antes. El cambio de contexto le permite a tu mente cambiar por completo y eliminar el contexto del área en la que se encuentra. Permite que tu mente se concentre en un nuevo contexto y una nueva situación.

Esto puede ser complicado en una oficina donde se espera de manera implícita que las personas deben permanecer en su espacio de trabajo para ser productivas. Y solo existen algunos “descansos biológicos” (descansos que están relacionados con la biología humana, también conocidos como ir al baño) que alguien puede tomarse en tales situaciones. ¿Cómo puedes darte esos cambios de contexto que tu mente necesita en tales situaciones?

También podrías ser capaz de obtener esos cambios de contexto apartando tu vista de la pantalla por unos momentos. Es una buena idea apartar la mirada de la pantalla de vez en cuando para dar a tus ojos un descanso. Dar a tu mente un descanso mientras das un descanso a tus ojos puede ser el incentivo para que realices ambos.

Cambiar de posición de sentado a estar de pie, también puede ser un buen cambio de contexto donde te permites un cambio en tu espacio físico de trabajo. Puede ser tan simple como levantarse y estirarse de vez en cuando, o tan complejo como levantarse y agacharse en tu escritorio. Diciéndote a ti mismo que hay dos contextos en tu escritorio: sentado y de pie, podría ser suficiente para darte un cambio de contexto y el descanso que tu mente necesita.

Si tu lugar de trabajo tiene una cultura en la que se te permite abandonar tu escritorio y cambiar de lugar, eso sería un gran cambio de contexto. El añadir un componente físico (tanto como seas capaz) a tu cambio de contexto puede ayudar a tu mente a relajarse y retomar energías.

Tendrás que experimentar con algunos de estos ejemplos para determinar qué te funciona a ti. Como mínimo querrás que tu mente sienta que no necesita estar alerta todo el tiempo. Quieres que tu mente se tome unos periodos de respiro entre sesiones de escribir código para poder eliminar los restos de esa sesión de tu “caché” mental y almacenarlos en la memoria a

largo plazo. Así cuando regreses a tu sesión de crear código, será más probable que recuerdes qué estaba pasando.

También es posible que descubras cuando te alejes de tu equipo por un momento, que olvidarás lo que estabas haciendo previamente. Eso está bien. Lo que recomendaría es mantener un diario o registro de en qué estabas pensando con tanto detalle como necesites. Ya sea escribiéndolo de manera física en un papel o usando un archivo de texto para tener suficientes pistas para continuar el trabajo donde lo dejaste.

5.4 Pensamiento productivo

Lo siguiente, deberemos darnos cuenta que la productividad no es una constante. Hay días en los que nos encontraremos generando niveles notables de código y código de calidad y días en los que tendremos suerte si podemos unir una cadena coherente de palabras para un comentario en el código. En el mismo día tenemos diferentes niveles disponibles de energía y enfoque mental. Depende de nosotros ser conscientes de estos niveles y comprender cual podría ser nuestra productividad para el día.

Entender estas variaciones de productividad puede permitirnos evaluar si el día nos permitirá o no generar código que necesita ser generado, pero hay un nivel inferior que creo que es importante.

Ponemos mucho énfasis diariamente en completar tareas y cumplir plazos. Este énfasis puede crearnos fuertes vínculos con la terminación de las tareas y los plazos. A veces esto es debido a factores externos (la “ruta crítica” del proyecto requiere que lo hagamos en una fecha y hora determinadas). Pero muchos de nuestros plazos son plazos internos que fueron establecidos por nosotros mismos. Nos marcamos una meta, de que seremos así de productivos al final del día. La condición no declarada de este plazo de productividad interna es que nos sentiremos culpables y avergonzados si no logramos el objetivo. Sentiremos que no estamos a la altura de nuestras expectativas y nos preguntaremos si somos dignos de la tarea que tenemos entre manos. Sentiremos que nuestro día ha sido desperdiciado y nos preguntaremos si somos capaces de hacer algo.

Es mejor para nosotros el eliminar los plazos cuando sea posible. No podremos eliminar los externos en los cuales los compañeros están esperando nuestras contribuciones (aunque es posible volver a negociar estos plazos si no son muy importantes) pero podemos dejar de lado el deseo de conseguir esos niveles arbitrarios de productividad y esos plazos arbitrarios.

Las metas arbitrarias pueden funcionar para algunas tareas. Algunos ejemplos de esto son los concursos de programación de juegos que solo duran una semana, lo que hace que los equipos se centren en las piezas críticas del diseño y la implementación de su juego para lanzarlo en el tiempo asignado. Este puede ser un ejercicio divertido para centrar esfuerzos, pero también conllevan mucho estrés y presión antes del fin de plazo del concurso. Si continuamente te sientes culpable e indigno porque parece que no puedes alcanzar las metas que te propusiste, entonces deberías reconsiderar si es útil usarlas.

Un truco que me ha ayudado es crear pequeños espacios de enfoque concentrado. Este truco se describe en la próxima sección.

5.5 Contenedores

Deberíamos reemplazar los plazos que no son estrictos (plazos que no se nos han impuesto de manera externa) con un compromiso de trabajar en un proyecto durante un periodo de tiempo. Un truco que he encontrado útil, es la idea de “contenedor de enfoque cronometrado”. Cuando realizo un contenedor de enfoque cronometrado empiezo escogiendo en qué me centraré durante el tiempo que durará ese contenedor. Una vez que está escogida la tarea a realizar, establezco un cronómetro en mi sitio de trabajo y después me centro en esa tarea con toda mi atención plena durante resto del tiempo que dure el cronómetro. He tenido buenos resultados usando 10 minutos pero sesiones de menos tiempo con 5 minutos o más largas como 30 minutos también pueden ser útiles. La tarea seleccionada al comienzo del contenedor es la única cosa en la que trabajo y trato de hacer lo posible para asegurarme de que no haya interrupciones (ya sean internas o externas) hasta que el contenedor sea completado. Cuando el trabajo se ha realizado

finalizo la tarea con lo que sea que haya completado, anotando cualquiera de las siguientes acciones para esa tarea en mi lista de próximas acciones y después me tomo un descanso rápido (unos 5 minutos) antes de comenzar con el siguiente contenedor. El siguiente contenedor puede ser una continuación de la misma tarea o puedo seleccionar otra tarea, pero la idea es simple: solo me centro en la tarea que tengo delante de mí durante el tiempo asignado. Cuando mi mente trata de divagar o tengo la tentación de "únicamente comprobar una cosa", entonces hago una pausa por un momento y determino si eso es realmente importante. La mayoría de las ocasiones no es tan importante y entonces hago una nota rápida para comprobar lo que sea cuando termine el contenedor.

Podemos utilizar estos contenedores para vencer nuestros deseos de realizar múltiples tareas al mismo tiempo. Solo nos enfocamos en una cosa cada vez. También podemos utilizar contenedores para permitir que la sesión fluya hacia donde quiera llevarnos. Cuando comenzamos un contenedor no lo comenzamos con la idea de finalizar una tarea en particular, en vez de eso vemos hacia dónde la sesión quiere llevarnos. No hay que judgar la calidad del trabajo en el contenedor, solo con la expectación de que trabajaremos durante la duración del contenedor. No hay expectativas por el trabajo que vayamos a realizar, solo que trabajaremos hasta que acabe el contenedor. Si completamos la tarea antes de que acabe el contenedor ¡eso es genial! Así podremos seleccionar la tarea para el siguiente contenedor. Si el contenedor acaba y todavía estamos en medio de una tarea, entonces podemos anotar qué nos queda y qué pasos vamos a dar para llegar allí. Entonces podemos seguir trabajando un poco más o podemos tomarnos un rápido descanso y después regresar al trabajo con un contenedor de enfoque nuevo.

El concepto subyacente del contenedor de enfoque cronometrado, es aceptar trabajar dentro de los límites del contenedor sin juzgar el trabajo realizado o el progreso realizado. Cuando realizamos el trabajo cerramos el contenedor reflejando qué hicimos y dónde necesitamos ir. Nos damos permiso para no preocuparnos en el momento por nuestro progreso, pero nos permitimos momentos donde poder revisar nuestro progreso y anotar cuánto ha progresado nuestro viaje. Nos permitimos la libertad de trabajar simplemente en el momento sin temor a juicios, represalias o auto

recriminaciones. El contenedor es un regalo de trabajo ininterrumpido (o al menos todo lo ininterrumpido que podamos gestionar) que nos damos. Nos hacemos un gran favor al cerrar otros programas, desactivar las notificaciones y darle a esta tarea toda la atención que merece.

Te invito a incorporar esta práctica de realizar contenedores focalizados cada día. Creo que son una excelente manera de darnos permiso para enfocarnos en una cosa a la vez sin la necesidad de preocuparnos sobre qué conseguiremos durante la duración del contenedor. Nos permite enfocarnos en una cosa a la vez y hacerlo ofreciendo lo mejor de nuestras habilidades. La limitación de trabajar en una cosa a la vez sin pensar sobre otros aspectos del trabajo que tenemos que hacer puede ser liberadora y espero que trabajar con estos contenedores te pueda dar una sensación de lo que se siente al realizar un trabajo completamente concentrado.

Todo este libro fue creado y editado utilizando esos contenedores de enfoque. Me llevó unos 10 minutos por contenedor escribir el borrador inicial y después utilicé contenedores de 10 minutos para editar el libro. Algunas veces se extendían hasta contenedores de 15 o 20 minutos pero fue porque estaba tan metido en la materia que no quería parar. Esto está en contraste con la forma en la que antes realizaba las tareas. Por lo general, necesito superar el obstáculo inicial de asignar media hora o más a la tarea. Esto significa que necesito sentir que tengo suficiente control sobre mi horario para reservar ese tiempo. Como no tiendo a sentir que tengo ese nivel de control sobre mi horario, tiendo a postergar la tarea. Con un contenedor de enfoque, pienso para mí mismo: “Puedo tomarme 10 minutos para trabajar en esto”, que es el tiempo suficiente para que mi mente no sienta que debería estar haciendo otra cosa. Con cada contenedor vi gradualmente cómo se desarrollaba el progreso de este libro. Eso luego retroalimentó mi deseo de seguir trabajando en este libro, lo que ayudó a reducir la fricción mental para seguir haciendo los contenedores para trabajar en el libro. Creó un ciclo de retroalimentación positiva en el que esperaba con ansias la próxima vez que pudiera hacer el contenedor y trabajar en el libro.

5.6 Distracciones

La vida está llena de distracciones. Muchas son las cosas que requieren nuestra atención y muchas de esas distracciones están fuera de nuestro control. Alguien llega a nuestro puesto de trabajo y necesita de nuestra atención en ese momento. Un tema que hemos tratado por correo electrónico que pensábamos que estaba resuelto se convierte en una discusión acalorada y reclama nuestra atención. Algo sucede en casa y ahora nuestra mente se divide entre preocuparnos por nuestras tareas laborales o preocuparnos por lo que sucede en casa. Cualquiera que sea la causa, hay momentos en que nuestra atención no está donde queremos y nos sentimos atraídos en todas direcciones a la vez.

Es aquí cuando los contenedores son más útiles. Si algo interrumpe el contenedor podemos determinar si es más importante que el trabajo que estamos realizando. Si determinamos que es más importante que lo que estamos haciendo en ese momento podemos detener el contenedor con la idea de que regresaremos al trabajo una vez que hayamos resuelto la interrupción. Si la interrupción no es más importante entonces podremos llegar a un acuerdo (ya sea con quien nos haya interrumpido o con nosotros mismos) en el que nuestro enfoque necesita estar aquí en el trabajo hasta que finalice el contenedor. Podremos darle a esa interrupción toda nuestra atención cuando el contenedor haya acabado. No necesitamos dividir nuestra atención entre el trabajo y la interrupción, en vez de eso le daremos a ambos toda nuestra atención a su debido tiempo.

Este método crea una delimitación simple entre nuestro trabajo y el resto del mundo, pero solo porque sea simple no quiere decir que sea sencillo. Mantener esa delimitación entre nuestro trabajo y el mundo que nos rodea puede ser un reto, especialmente si la cultura en la que te mueves es la de resultados inmediatos.

No tengo buenas noticias para ti si la cultura en la que trabajas demanda tu atención todo el tiempo. Lo mejor que puedo ofrecer es un sucedáneo de contenedor que al menos te pueda ofrecer algunos periodos de concentración sin distracciones. Si te sientes siempre en tensión porque algo podría ocurrir en cualquier momento, continuarás siendo menos efectivo a diferencia de si pudieras silenciar el mundo por un momento. También te reto a examinar si esa sensación es realmente cierta ¿estás siendo

constantemente siendo acechado por interrupciones? Probar esa teoría puede ser una buena práctica. Mantén un registro (ya sea en una hoja de papel, un archivo de texto, una hoja de cálculo o una base de datos, eso depende de tus gustos) de cuando hiciste un contenedor de atención y si ese contenedor fue interrumpido o no. Si al final encuentras que está siendo más veces interrumpido que no, entonces necesitas determinar qué es lo que está causando las interrupciones y evaluar si es algo que puedes controlar. Hay muchas maneras de manejar y minimizar las distracciones en el lugar de trabajo en las que no voy a entrar aquí, pero ser consciente de las distracciones y determinar de dónde vienen será la clave para saber cómo mitigarlas en el futuro.

Se también consciente de las distracciones auto impuestas que has añadido a tu vida. ¿Necesitas la notificación inmediata de que a alguien le ha gustado algo que compartiste? ¿Es la anécdota tan divertida como para cambiar tu contexto actual para así poder compartirla con tus amigos o colegas? ¿Realmente necesitas algo que emerja de repente en tu campo visual, para saber que tu reproductor de música ha cambiado de canción? ¿Estarías dispuesto a sacrificar tu atención y tu flujo de trabajo en el día a día porque un programa detectó un cambio en tu entorno, independientemente de la importancia de ese cambio?

Añadimos estas distracciones en nuestras vidas porque nos preocupa que nos perdamos algo importante. Los programas también vienen configurados con sus notificaciones activadas, así el usuario puede recordar el estado del programa todo el tiempo. Quizás es útil, pero para mí es muy molesto. En mi vida laboral me he sentado en escritorios de otras personas y me he sorprendido por el número de notificaciones que recibían en el periodo que estuve allí (en un lapso de diez minutos o incluso menos). He visto a personas interrumpir su línea actual de pensamiento porque una notificación de un mensaje de una cosa que nada tenía que ver con la tarea actual que desarrollaban les distraía. ¿Qué pasó con el pensamiento original? Tienen que regresar mentalmente a él y recordar dónde lo habían dejado, normalmente eso conlleva un gran esfuerzo mental.

Te reto a desactivar todas las notificaciones que puedas y disfrutar de la experiencia que es estar sin ellas. Eso puede ser tan simple como cerrar una

aplicación cuando has acabado con ella o puede ser tan complejo como cambiar la configuración de una aplicación para que no te notifique cuando llegue un mensaje. Necesitarás probar este método y ver qué es lo que mejor funciona con tus necesidades y concentración. Una buena regla general es “¿qué es lo que comprueba esta cosa que es lo suficientemente importante como para dejar mi importante trabajo y concentrarme en esta cosa?”. Si puedes valorar tus notificaciones para que solo las notificaciones más críticas te lleguen en el momento adecuado entonces disfrutarás de un mayor relax y concentración en tu trabajo. Así no tendrás que analizar la notificación para determinar si lo que estás viendo es importante o no.

Una de las razones que he escuchado de compañeros para mantener activas sus notificaciones es que sienten que podrían recibir algo que requiriera una respuesta inmediata. Hemos creado una cultura donde sentimos la necesidad de responder a los mensajes en el momento que los recibimos. Me atrevería a decir que la mayoría de los mensajes que recibes durante el día no requieren de la atención que les estás dando y seguro que tampoco el nivel de atención que justifique la interrupción que realizas para verlos y responderlos. Sería mejor programar varios periodos del día en el que no hagas otra cosa que comprobar y responder a los mensajes. Organiza esto con la frecuencia más baja como puedas. Algunas personas recomiendan 2 o 3 veces al día, pero incluso estableciendo un límite donde compruebes tus mensajes una vez a la hora, puedes conseguir una gran mejora frente a la cantidad de veces que actualmente compruebas tus mensajes. Necesitarás judgar por ti mismo la frecuencia con la que comprobar tus mensajes basándote en tus necesidades y en tu ambiente de trabajo. También considera la persona a la que estás respondiendo. ¿Tiene sentido darle a esta persona una respuesta rápida y semi-pensada, o este mensaje requiere más tiempo para darle vueltas en tu mente antes de responder? Darte tiempo para pensar en la respuesta puede brindarte información adicional sobre un problema que podría no ser evidente en el momento. Esto podría significar la diferencia entre una respuesta bien pensada frente a una avalancha de tormenta de ideas de ida y vuelta a medio pensar utilizando tu aplicación de mensajería. Responder a todo inmediatamente es muy estresante y requiere una gran cantidad de atención que podría utilizarse mejor en tu trabajo de programación.

Puede parecer desafiante y extraño vivir sin notificaciones y sin la necesidad de responder a cada mensaje y notificación, pero nuestra atención es finita y limitada. Mantener el enfoque a lo largo del día es desafiante y estresante. Si podemos limitar la cantidad de distracciones que recibimos a lo largo del día, nos daremos la libertad de no tener que trabajar tanto para mantener nuestra atención en sintonía con nuestras tareas de programación. Podemos decir “ahora no” a nuestras distracciones y resolverlas en un momento más apropiado.

6 El mapa no es el territorio

6.1 El panorama cambiante de la programación

La única constante en el campo de la programación, es que siempre está en constante cambio. Los lenguajes de programación llegan a tener mucho renombre y después desaparecen con el paso del tiempo. Lo que una vez fue tomado como un hecho ahora está considerado obsoleto (o incluso “considerado dañino” como señalan algunos ensayos).

Cuando me gradué en la universidad, aprendí Pascal, Modula2 y Ada. Desafortunadamente esos lenguajes empezaron a decaer en popularidad en favor de C. Cuando comencé en mi primer trabajo de programación como “profesional”, Perl era el lenguaje de elección (en parte porque Perl podía ser transformado de manera sencilla en scripts CGI de la época y era considerado superior a herramientas para realizar scripts como `awk` y los tradicionales scripts para la shell). En el momento de escribir esto estoy utilizando Python como mi lenguaje de desarrollo principal y preveo que tendré que buscar otros lenguajes de programación para expandir mi carrera como programador.

Programar requiere flexibilidad. Es difícil aprender solo una manera de hacer las cosas y mantener eso de manera relevante durante 20 años. Recuerda cual era la tecnología actual hace 20 años y no tendrás ninguna duda en darte cuenta que las cosas eran bastante diferentes entonces. Si quieres poner en práctica un ejercicio divertido, busca artículos describiendo la tecnología puntera de hace 20 años y ver en ellos cuanto de aquello reconoces.

6.2 Aprender a aprender

Aprender metodologías y tecnologías específicas no es una buena estrategia a largo plazo para los programadores. Es más recomendable si aprendemos a aprender y, lo que es más importante, cómo aprendemos nosotros mismos. Eso suena simple: una vez que hayamos descifrado cómo aprender de manera efectiva, seremos programadores efectivos. Desafortunadamente, no existe una manera infalible de aprender que funcione para todas las personas. Personas diferentes, aprenden de maneras diferentes. Todos tenemos estilos de aprendizaje que funcionan mejor cuando se enfatizan ciertas cosas. Algunas personas aprenden mejor en el aula de una clase, mientras que otras aprenden mejor con el estudio auto dirigido (libros, grabaciones de vídeo, etc.). Algunas pueden leer un libro y entender perfectamente la materia, mientras que otras pueden necesitar enfoques más visuales. Si tienes el lujo de poder probar varias metodologías diferentes para aprender, te animo a que uses todas las que puedas para descubrir cuál funciona mejor en tu caso. Comprender lo que funciona para ti, será clave para ayudarte a progresar y crecer.

He encontrado que algunos principios sencillos funcionan mejor en mi caso. El primero es la repetición. Aprendo mejor cuando hago algo de manera diaria, una y otra vez, en pequeñas dosis. La segunda manera es teniendo una pequeña meta que pueda conseguir. Así que para mí tener una práctica diaria en un proyecto donde pueda trabajar y conseguir un objetivo es lo que mejor me funciona. Cuando estaba aprendiendo Python me uní a la PyWeek, una competición de una semana de duración de programación de un juego, donde el tema a tratar se anuncia al comienzo y toda la programación transcurre en una semana. Durante esa semana entera encontré tiempo para completar mi juego y al final de la semana había aprendido más sobre Pygame (la biblioteca que utilicé para mi juego) y Python que lo que había conseguido en semanas anteriores al PyWeek. Realizar una semana de duración de esta *game jam* (como lo llaman) es un poco extremista, pero me dio un objetivo claro (un juego funcional completo) y un tiempo dedicado para conseguirlo (una semana). Con el paso del tiempo he aprendido más sobre Python con varios proyectos (tanto profesionales como personales) que tenían una práctica diaria y objetivos claramente definidos.

Necesitarás experimentar para ver qué es lo que mejor funciona en tu caso. El principio que subyace es que el proceso de aprender debería ser algo que tu puedas utilizar para cualquier lenguaje o concepto en programación. También debe ofrecer la menor cantidad de resistencia a su aprendizaje. Tu capacidad para aprender y adaptarte será vital para tu experiencia como programador, así que entender tu propio proceso de aprendizaje y qué funciona mejor en tu caso te ayudará en este proceso.

Como mínimo, reserva 10 minutos al día para un contenedor (consulta el capítulo anterior) para una lectura y el aprendizaje enfocados. Hay mucho que aprender en programación y crear un hábito de aprendizaje te ayudará a mantenerte al día. Recuerda, sin embargo, mantener tu aprendizaje contenido en pequeños fragmentos. Mucha información puede abrumarte y hacerte pensar que no puedes aprenderlo todo. Tienes razón, no puedes aprenderlo todo de una sola vez. Si alguien te dijera que bebieras uno de los Grandes Lagos de una sola vez, sería difícil completar la tarea (nota: ¡no intentes esto!). Sin embargo, si llenaras un vaso de agua varias veces al día de uno de los Grandes Lagos y lo bebieras (10 minutos cada vez), comenzarías a hacer una mella apreciable en la reducción de ese lago a lo largo de tu vida. (Claro, puede que no parezca mucho desde el exterior, pero ese es el cruce donde la realidad y las metáforas se rompen).

Cada día tienes una oportunidad de aprender sobre más sobre ordenadores y su programación. Realizar una pequeña parte cada día para aprender un poco más, te ayudará en tu viaje.

6.3 Cómo escoger qué aprender

Hay muchas oportunidades para aprender, ya sea mediante libros, tutoriales, vídeos o cursos a distancia con tu ordenador. También hay una gran variedad de temas sobre los que aprender. ¿Cómo decides qué es lo más importante para aprender? ¿Cómo gestionas lo que estás aprendiendo? ¿Cómo conseguir evitar sentirse abrumado con las opciones disponibles?

Esto nos vuelve al tema de centrarse en una cosa cada vez y entender cómo aprendes de una manera eficiente. Esta retroalimentación te ayudará a

decidir qué será lo próximo a aprender. Una manera puede ser pensando en las cosas que más te apasionan ahora mismo, ¿qué te motiva en este momento? Si hay algo por lo que estás ansioso por aprender empieza por eso. Si tienes múltiples cosas que te apasionan o te interesan entonces colócalos en una lista y observa si estás más interesado en unos temas que en otros. Si aún tienes problemas para decidir de los temas de la lista entonces escoge uno al azar (lanza un dado o crea un generador de números aleatorios para seleccionar uno, eso podría ser en sí mismo un proyecto).

Si tienes problemas para pensar en algo para aprender y estás luchando para encontrar un elemento que te resulte interesante, entonces date permiso para navegar por la red y encontrar qué hay disponible. Observa las conversaciones de otros programadores y descubre de qué están hablando. Acude a una reunión de programadores para seguir las discusiones de lo que están hablando. O, si realmente estás atascado, explora algunas listas de ofertas de trabajos para averiguar qué es lo que buscan las personas que ofrecen trabajo y observa si algo de eso te despierta algún interés.

Esto no trata de escoger lo más útil o la cosa más importante, aunque tu situación actual puede hacer que algunos temas tengan más relevancia sobre otros, se trata de averiguar qué capta tu atención y dónde enfocarte. No te preocupes con realizar la elección perfecta que te proporcionará tu siguiente trabajo o impulsará tu carrera. Este ejercicio trata sobre hacer una elección para aprender algo interesante y que te enganche el tiempo suficiente para aprender más sobre ello.

Una vez que hayas escogido qué quieres aprender es el momento de centrarte en aprenderlo. Si tienes una metodología preferida (libros, vídeos, tutoriales, clases, etc) entonces dedica un tiempo (no más de una hora) a buscar qué recursos hay disponibles. Algunos temas tienen recursos disponibles para los principiantes que enumeran las cosas que la comunidad cree que son útiles para los programadores que están empezando, mientras que para otros puede ser necesario preguntar a la comunidad por dónde comenzar. A menudo algo tan simple como un tutorial puede ser una buena manera de comenzar con este ejercicio.

¡Si puedes encontrar algunos recursos en un corto periodo de tiempo será genial! Comienza tu proceso de aprendizaje con esos recursos. No te

preocupes si son los recursos adecuados o quizás te lleven por un camino equivocado, simplemente comienza con ellos y ya los evaluarás más tarde. Por ahora es más interesante simplemente comenzar.

Una de las trampas en la que soy culpable de caer, es en tratar de encontrar los mejores recursos para aprender un tema. Paso horas buscando el libro perfecto, los vídeos adecuados, los cursos idóneos, lo que sea, trato de encontrar los mejores materiales disponibles. Quiero reducir la cantidad de comienzos en falso cuando aprendo de un tema. Esto que parece una una búsqueda correcta (después de todo, ¿por qué no ibas a querer los mejores materiales que haya disponibles?) es una trampa que puede llevarte a pasar más tiempo buscando sobre cómo estás aprendiendo en vez de estar ya aprendiendo. Incluso peor, si el material comienza a confundirte (lo que es altamente probable cuando estás aprendiendo algo nuevo) pasarás tu tiempo de aprendizaje pensando si escogiste la decisión adecuada escogiendo este material. Te preguntarás si escogiste el material idóneo y continuarás buscando el mejor material (quizás esas buenas y excelentes críticas realmente no sabían después de todo de lo que estaban hablando). Esto disminuye tu habilidad para aprende sobre el tema porque estás más enfocado en tratar de discernir la calidad de lo enseñado y no en dedicar tiempo en la enseñanza actual.

Después de unos días de sesiones prácticas date la oportunidad de comprobar y ver cómo estás aprendiendo. ¿Te sientes atraído o no estás disfrutando de esto? Si no te sientes atraído (el material no está organizado, el instructor es confuso, los ejemplos no funcionan, el material asume que ya estás familiarizado con otro tema, etc) entonces date permiso para buscar un material mejor o un tema diferente que te interese más. Incluso si la experiencia de aprendizaje no fue óptima, tendrás una mejor idea de lo que buscar cuando escojas algo nuevo. Tendrás un conocimiento de cuales son tus lagunas con este tema y tendrás un mejor conocimiento de lo que estás buscando en los materiales de aprendizaje.

Si encuentras que el tema que estás tratando de aprender ya no te interesa, date un momento para reflexionar sobre el motivo de esto. ¿Es un tema complicado? ¿Te sientes preparado para ese tema? ¿Estás actualmente sobrecargado con otros proyectos y te sientes cansado para abordar este

tema? A veces creemos que estamos preparados para aprender sobre un tema, solo para darnos cuenta que hay algo más que necesitamos saber antes de poder entender por completo ese tema. Está bien buscar recursos adicionales y enfocarte en ellos antes de abordar este tema. Simplemente se consciente de tus luchas y tu diálogo interno. Se honesto contigo mismo sobre el motivo por el que quieres cambiar a algo diferente. Obsérvate en la dificultad y se consciente si estás queriendo huir porque es difícil o si realmente no estás preparado para ello o no estás interesado en este tema. Observa si puedes ahondar más con la dificultad y siente cuándo comienzas a sentirte sobrepasado por ella. Date permiso para apegarte a la dificultad tanto como puedas y presta atención a tus sentimientos e impulsos mientras practicas con ella.

Trata tu aprendizaje como un proceso iterativo, con periodos regulares en los que comprobar tu progreso. Piensa en cómo te sientes cuando estás aprendiendo. ¿Te sientes emocionado y comprometido o te sientes cansado y retraído? ¿Procrastinas cuando piensas en este tema? ¿Cuando te centras en tu aprendizaje tu mente divaga? Se consciente de estos sentimientos a medida que ocurren durante la sesión centrada y reflexiona sobre ellos cuando pienses en tu proceso de aprendizaje. Más tarde puedes reflexionar sobre estos sentimientos y ver los patrones en tu proceso de aprendizaje. Si te sientes cansado mientras estás aprendiendo podrías intentar cambiar cuando realizas tus sesiones de aprendizaje. Quizás necesites dormir más o necesites encontrar otros materiales que sean más estimulantes. Si te sientes abrumado quizás necesites empezar con algo más básico antes de afrontar este proyecto más difícil. Si estás confundido quizás haya alguien a quien puedas plantearle preguntas para obtener más claridad sobre el tema. Estas respuestas pueden no ser evidentes mientras estás en viviendo ese momento (quizás estás muy ocupado sintiéndote frustrado para entender de dónde viene esa frustración), pero con la práctica tendrás más recursos para observar tus sentimientos. Cuando eres consciente de estos sentimientos podrás usarlos para aprender cómo funciona tu mente y entender qué necesita para mantenerte comprometido con tu aprendizaje.

6.4 Resistencia y el contenedor

Cada vez que aprendemos cosas nuevas nos colocamos en un lugar vulnerable e incómodo. Tomamos las cosas con las que estamos familiarizados y las aplicamos a medida que nos adentramos en nuevos territorios. Nos volvemos inseguros con el resultado, ¿será un éxito o será un fracaso? ¿Será este tema difícil de aprender? ¿Nos ayudará o nos perjudicará? ¿Hemos escogido la opción equivocada para aprender y eso nos costará oportunidades a largo plazo?

La incomodidad y la incertidumbre son sin duda una parte del aprendizaje, pero en vez de pensar en ellas como algo a ser evitado pensemos en ellas como faros. Unas balizas que nos indica el camino y nos iluminan cuando estamos en territorios desconocidos. Cuando nos sentimos inseguros sobre lo que estamos haciendo, ese sentimiento significa que estamos entrando en un nuevo territorio. En vez de evitarlo o desear comodidad, podemos disfrutar por estar en un territorio desconocido y sentir esas breves punzadas de miedo y duda. Podemos decir: “Voy a aprender algo nuevo. Estoy asustado y no sé dónde me llevará esto, pero está bien. Estoy dispuesto a ver a dónde lleva esto y disfrutar del viaje.”

Estamos condicionados a pensar en lo desconocido como algo a lo que temer. Estas emociones nos han sido útiles. Nos han impedido aventurarnos demasiado de nuestra zona de confort y explorar lo desconocido. Cuando vives en un bosque o en cuevas, lo desconocido puede albergar todo tipo de peligros. Tiene sentido no provocar esos peligros apareciendo delante de sus puertas. Pero programar no es lo mismo que aventurarse en un bosque oscuro o asomarse a una cueva húmeda, programar apenas conlleva la cantidad de peligro que le otorgamos. En vez de eso, debemos darnos cuenta que no estamos en ningún peligro mortal. Nuestros miedos simplemente nos están haciendo saber que estamos aventurándonos en los territorios inexplorados de la ignorancia. Depende de nosotros dar a conocer a nuestros miedos que esto está bien y que al explorar estos territorios solo encontraremos conocimiento.

Steven Pressfield en *La guerra del arte* denominó a estos sentimientos como “Resistencia”. Considera la Resistencia como una especie de ser mitológico que vive en cada uno de nosotros para frustrar los actos creativos. A medida que el trabajo progresa la Resistencia aumenta la

presión para detenernos introduciendo los sentimientos de miedo y ansiedad que he mencionado anteriormente. Pienso en la Resistencia como algo que también ocurre cuando estamos aprendiendo, especialmente si estamos aprendiendo sobre herramientas que nos ayudan en nuestras actividades creativas. Pressfield limitó su definición a las personas creativas que estaban trabajando para finalizar un trabajo creativo (libros, pinturas, juegos, etc.) pero yo estoy expandiendo su definición al propio proceso de aprendizaje. En nuestro caso la Resistencia aparece cuando estamos aprendiendo las herramientas que nos ayudarán a ser más creativos. La Resistencia es lo que nos dice que no somos suficientemente buenos para aprender esas cosas, o cuando estamos inseguros sobre los beneficios que nos acarreará. Trata de mantenernos seguros con lo que ya conocemos.

Este es el motivo por lo que el “contenedor focalizado” es tan importante: nos da pequeñas dosis de incomodidad y dificultad en porciones manejables. Podemos guiarnos a través de pequeñas cantidades de incomodidad diarias y seguir aprendiendo a pesar de nuestra incomodidad. Nos ayuda a trabajar a pesar de nuestra tendencia a evitar y ocultar situaciones difíciles. Si nos centramos en una sola cosa cada vez podemos mantenernos apartados de las distracciones sobre si esta es o no la cosa en la que deberíamos estar trabajando. Lo que sea en lo que estemos trabajando en este momento es exactamente en lo que deberíamos estar trabajando. Cualquiera que sea el material de aprendizaje que tengamos delante es lo que deberíamos estar aprendiendo. Podemos estar seguros de saber que todo lo que estamos haciendo durante la duración del contenedor es exactamente lo que debería de ser. Cuando finalice el contenedor podemos volver a evaluar cómo fue y qué retos se presentan por delante.

6.5 Trazado de objetivos a largo plazo

A medida que progresas en tu proceso de aprendizaje empezarás a ver que muchas de las cosas que llamamos programación están interconectadas. Los lenguajes de programación se prestan muchas cosas unos a otros y las ideas que parecían nuevas e innovadoras tienen sus raíces en conceptos que se remontan a los orígenes de la computación. En vez de disuadirnos, esto

debería animarnos a abrir las puertas de la programación aprendiendo conceptos simples y transferibles. La pregunta es ¿cuales son?

La respuesta más simple es “todos ellos”, pero eso es difícil de satisfacer o imposible. Una respuesta menos descarada sería “los suficientes para empezar a ver cómo surgen patrones” pero eso suena más a una idea de perogrullo que a algo que podamos utilizar para comenzar a realizar nuestras metas a largo plazo para el aprendizaje.

En vez de darte un consejo específico sobre qué conceptos te serán más útiles en tu búsqueda de llegar a ser un programador mejor, voy a sugerirte una técnica que podría ayudarte a identificar lo que podría ayudarte.

Los lenguajes de programación mencionarán conceptos que comparten. Cada vez que estés aprendiendo y veas una mención a alguno de estos conceptos, haz una nota de ello y siguen enfocado en lo que estás aprendiendo ahora. Cuando hayas completado el periodo de aprendizaje diario, revisa la lista de estos otros conceptos y haz una búsqueda para ver qué muestran. Si hay otras cosas que aparecen entonces escríbelas en tu lista. Estos conceptos podrían no tener sentido en el momento pero tener una lista disponible y a los que se refieran podría ayudarte a hacer conexiones sobre programación que podrías no haber notado de otra manera.

Cuando aprendí JavaScript me dí cuenta que alguien mencionaba que JavaScript derivaba de otros lenguajes como Scheme. Scheme es un lenguaje funcional basado en Lisp y que fue creado como lenguaje de enseñanza para programación funcional y recursividad. Así que tomé un pequeño rodeo para aprender Scheme, personalmente porque me era más interesante que JavaScript. Llámalo “procrastinación activa”, si te sientes benévolo. Lo que aprendí mientras aprendí Scheme, motivó mi interés en otros lenguajes funcionales y la programación funcional. Esto a cambio me ayudó a entender algunos paradigmas de la programación funcional que se estaban convirtiendo muy populares en Python (listas de comprensiones, lambdas, etc). Al tomar un breve rodeo en mi aprendizaje de JavaScript aprendí más sobre toda una familia de lenguajes y ahora siento que entiendo JavaScript y Python con una mayor claridad que cuando empecé.

No estoy sugiriendo que todo el mundo debería seguir los pasos de la “procrastinación activa” como hice yo (mientras escribo esto, todavía estoy en el proceso de aprender JavaScript), pero ayuda el realizar notas de los conceptos que vas encontrando y ahondar un poco más en ellos.

Esta es una manera de trazar los objetivos del aprendizaje (darse cuenta de las otras conexiones que aparecen mientras estás aprendiendo y ser curioso sobre cómo interactúan entre ellas), pero quizás necesitas una técnica diferente. Quizás estás bajo presión para aprender algo que te mantenga en el mercado laboral o necesites adquirir algún conocimiento para tu trabajo que necesita ser aprendido rápidamente. ¿Cómo trazas esas metas?

La presión de aprender rápidamente puede hacer que cualquier tarea parezca insuperable, especialmente si no sabes cuál es la mejor manera de proceder. Es posible que sientas la tentación de apresurarte en este proceso y confiar en retener el conocimiento que has aprendido. Este enfoque no conduce a la comprensión, conduce al estrés y al agotamiento. El enfoque que estoy esbozando está diseñado para ayudarte a aprender cómo aprender. La mejor forma de aprender algo rápidamente es entender cómo encajan otros conceptos con lo que estás aprendiendo. Esto es genial cuando tienes experiencia con muchos lenguajes y conceptos diferentes, pero para aquellos que aún no tienen mucha experiencia, se sentirán como si estuvieran tratando de empujar un elefante a través de un pequeño embudo. Aquí es donde te ayudará practicar el aprendizaje todos los días. Te ayudará a dividir los objetivos de aprendizaje más grandes en partes más pequeñas y te ayudará a reconocer el miedo y la incomodidad por lo que realmente son: el reconocimiento de que estás expandiendo tus habilidades a un nuevo territorio.

Las metas a largo plazo son simplemente metas que deben ser divididas en metas a corto plazo. Enfocarse en metas a corto plazo y permitirte corregir el rumbo y seguir algunas conexiones cuando sea necesario.

6.6 Fracaso y aprendizaje

Una de las cosas a las que tememos cuando estamos aprendiendo, es al fracaso. Nos preocupamos porque no aprender el tema rápidamente o al completo. Escogemos material que comienza de manera simple pero luego se vuelve muy complejo, y luchamos por mantener el ritmo. Intentamos escribir código de ejemplo en nuestros editores y nos encontramos necesitando ayuda para que funcionen. No logramos comprender el material y nos preguntamos si alguna vez aprenderemos lo que estamos tratando de aprender.

El fracaso es parte del aprendizaje. Si ya conocieras la materia no estarías aprendiendo.

Una de las razones para aprender practicando utilizando los contenedores, es porque así nos damos esos breves momentos de fracaso y repetición. La repetición es como mejoramos en cualquier cosa que estemos aprendiendo. El fracaso nos permite corregir el curso de nuestro aprendizaje para que podamos determinar la mejor manera de abordar esto la próxima vez que hagamos un intento.

A menudo sentimos que el fracaso es algo que debemos evitar, pero mientras estamos aprendiendo es inevitable. Nuestro proceso de aprendizaje requiere que fallemos para mejorar en lo que estamos aprendiendo. Ese es el objetivo del aprendizaje: volver a moldear nuestros cerebros para que finalmente puedan comprender los conceptos que estamos tratando de aprender.

Parte del aprendizaje es tener la mentalidad correcta para aprender. En lugar de sentir que estás constantemente fallando y luchando por mantenerte al día, es posible que desees abordarlo con una perspectiva diferente. En lugar de pensar “No puedo hacer esto. Es demasiado difícil”, acércate a ello con una mirada más curiosa “Todo esto es nuevo para mí. Es por eso que estoy practicando para aprender esto”. Darte una mentalidad más positiva te ayudará a evitar que te rindas cuando tengas problemas con el material.

6.7 Callejones sin salida y topografía cambiante

A veces nos encontraremos a nosotros mismos aprendiendo algo que es un callejón sin salida. Observamos nuestro progreso y no vemos una mejora real. No encontramos el tema tan atractivo o excitante como habíamos imaginado. Nos damos cuenta que lo que estamos aprendiendo es un callejón sin salida evolutivo en el campo de la programación. ¿Ahora qué?

Parte de nuestro proceso de aprendizaje es entender que nuestras expectativas sobre cómo algo evolucionará pueden ser completamente diferentes a cómo las cosas evolucionan de verdad. Visionamos toda clase de recompensas y clichés que nunca llegan. ¿Significa eso que estamos en un callejón sin salida? Yo creo que no. Podría suceder que lo que esperábamos que íbamos a hacer con nuestro recién adquirido conocimiento no está funcionando. Podríamos encontrar que nuestras expectativas sobre lo rápido que íbamos a aprender el tema no se están cumpliendo. También incluso podríamos esperar que nuestra carrera profesional se iba a ver respaldada por el tema de aprendizaje, pero el mercado de trabajo aún no ha reconocido nuestras recién adquiridas habilidades con ofertas de trabajo o más dinero.

Nuestro compromiso está relacionado con nuestras expectativas. La programación requiere una cierta cantidad de diversión y recompensa y si no encontramos la experiencia divertida y gratificante entonces es poco probable que queramos continuando seguir aprendiendo sobre ese tema. Nuestras mentes están esperando algo más que nos atraiga y comenzamos a anhelar cualquier otra cosa en vez de continuar con el proceso de aprendizaje. Después de todo ¿no deberíamos estar disfrutando de esto? Si no hay compromiso y no disfrutamos, entonces el aprendizaje se convierte en una losa. Nos distraemos más fácilmente mientras estamos intentando aprender y nuestras mentes se dispersan en vez de enfocarse en nuestra experiencia de aprendizaje.

También está el problema de aprender sobre temas que son callejones sin salida evolutivos. El mundo de la informática está lleno de restos de tecnologías y metodologías que ya no son relevantes o están consideradas “pasadas de moda”. Lo que una vez fue algo vanguardista ahora está considerado moribundo y la comunidad creada alrededor de esa tecnología o metodología se pasa a nuevas tecnologías o metodologías y abandona su

trabajo previo como un pueblo fantasma tecnológico. Cuando mencionamos que estamos aprendiendo sobre esos temas, caen sobre nosotros miradas curiosas de otros desarrolladores: “¿Por qué aprender sobre eso? Hemos cambiado a esta otra cosa”. Es como si hubiéramos oído que hay una fiesta y cuando llegamos a la fiesta, solo vemos a las personas que están recogiendo la basura y desmontando la mesa y las sillas. Nos sentimos como si nos hubiéramos perdido la mejor parte y nos preguntamos si tiene sentido seguir hacia adelante o encontrar otro tema.

Todo esto puede plantear sus propios problemas para aprender, pero depende de nosotros tener una visión más crítica de por qué empezamos todo este proceso de aprendizaje. ¿Qué nos trajo hasta aquí?

En cada uno de estos casos trajimos nuestras expectativas de cómo progresaría el aprendizaje. Trajimos la expectativa de que siempre sería divertido, atractivo y relevante. A veces nuestras expectativas de aprendizaje se cumplen, pero cuando no lo hacen nos sentimos sin ánimos y decepcionados.

En vez de sentirnos incómodos sobre cómo nuestras expectativas de aprendizaje con esta tecnología o metodología no se están cumpliendo, podemos adoptar un enfoque más consciente. Nos podemos observar a nosotros mismos en nuestros momentos de aprendizaje y notar si estamos tratando de aportar más que únicamente una atención focalizada al contenedor de aprendizaje. Podemos ser conscientes de que aprender está relacionado con cambiarnos a nosotros mismos y el cambio no siempre es divertido, atrayente o placentero. Podemos dejar a un lado nuestras expectativas y concentrarnos en el propio aprendizaje.

Eso no significa que no debamos ser conscientes de nuestros sentimientos. Sin duda deberíamos reconocer sentimientos como aburrimiento, ansiedad, desilusión, etc. Pero también deberíamos ser conscientes de dónde se originan esos sentimientos. ¿Estamos realmente aburridos o es solo nuestra mente tratando de decir que debemos parar para así poder hacer algo más divertido? ¿No nos apasiona este material porque no lo encontramos relevante o nos estamos simplemente distrayendo? ¿Es esto realmente un callejón sin salida en nuestro aprendizaje o simplemente nos estamos sintiendo atascados? Identifica cuándo aparece el sentimiento y se curioso

sobre lo que originó el sentimiento. Nota cuándo obtienes ese sentimiento y dónde lo sientes más en tu cuerpo. Permanece con ese sentimiento durante unos segundos y sigue sintiéndolo. Después, continua tu trabajo. Mientra estás trabajando sigue notando todos esos sentimientos que estás teniendo y repite el proceso de permanecer y notar tus sentimientos. Cuando acabes puedes reflexionar más sobre esos sentimientos y determinar qué indican esos sentimientos. A través de este proceso puedes determinar qué está causando estos sentimientos y notar si son solo una resistencia a aprender nuevo material o un deseo de escaparse a las distracciones o a algo más familiar.

Sin embargo, si te das cuenta que realmente no estás disfrutando mientras estás aprendiendo sobre ese tema, si sientes que estás invirtiendo más tiempo en auto convencerte de aprender que realmente aprender, entonces necesitarás tener una discusión honesta contigo mismo sobre porque estás aprendiendo sobre este tema. ¿Es el tema todavía relevante para ti o el tema se ha vuelto irrelevante? ¿Estás aprendiendo sobre esto por una obligación auto impuesta o impuesta por otras personas y esa obligación todavía está presente? ¿Estás tratando de aprender lo que sea porque estás preocupado por quedarte desfasado personal o profesionalmente? Reflexiona sobre lo que te indujo a comenzar a aprender sobre ese tema y determina si la situación ha cambiado. Si alguien se te acercara y te preguntara si te gustaría aprender sobre ese tema en los próximos días ¿lo considerarías?

Necesitarás reconsiderar tus verdaderas motivaciones para aprender sobre el tema y ver si todavía encajan con lo que quieres hacer con tu profesión como programador. También necesitas ser honesto contigo mismo sobre el motivo por el que estás aprendiendo este tema y por qué es importante para ti. Hay muchas cosas para aprender que son grandes salidas profesionales, pero si no tienes interés sobre el tema o solo estás aprendiendo “para que te contraten”, entonces tendrás más dificultades para aprender el tema que si lo hubieras escogido por un interés genuino por él. También tendrás que determinar si esto es simplemente una resistencia al aprendizaje. Tu reto será determinar tus verdaderos sentimientos sobre el tema y descubrir si verdaderamente has perdido el interés o simplemente estás luchando contra él.

Ha habido muchos temas en mi carrera profesional que he tratado de aprender, pero han sido muchos más sobre los que no he aprendido. Parte de los motivos por los que no los he aprendido es porque el panorama informático cambió mientras los estaba aprendiendo. En la escuela aprendí el lenguaje Pascal. Era razonablemente bueno en ello, pero con el tiempo mis conocimientos de Pascal fueron desapareciendo. Ahora mismo existe una escasa demanda de programadores competentes de Pascal, así que haber continuado desarrollando mis conocimientos de Pascal hubiera sido simplemente para mi propio disfrute. Encuentro que hay otros temas relacionados con la informática más interesantes, así que mis conocimientos sobre Pascal permanecen inactivos. Si Pascal emergiera de su estado moribundo, podría volver a retomar la decisión de reforzar mis conocimientos sobre Pascal, pero por ahora estoy contento por haber tomado la decisión correcta. En un punto de mi carrera profesional el lenguaje Java saltó a la fama. Pasé muchas sesiones aprendiendo Java hasta que me dí cuenta que no me gustaba el lenguaje. Me pareció muy engorroso y las direcciones que tomó no fueron las que quería seguir. Así que después de algunas reflexiones abandoné el aprendizaje de Java. ¿Fue todo esto tiempo perdido? Durante mis sesiones aprendí más sobre la Programación Orientada a Objetos y cómo los objetos encajan entre ellos. Aprendí más sobre recursividad mientras estaba tratando de resolver un problema para uno de mis proyectos. Estos conocimientos trascienden a Java, así que cuando empecé con Python fui capaz de transferir mis conocimientos sobre cómo funcionan los objetos de Java a Pytnon. Utilicé ese conocimiento para entender qué estaba haciendo Python y cómo era de diferente respecto a Java. Si surge la necesidad puedo repensar mi decisión de dejar de aprender Java y ver si esto me interesa otra vez.

Está bien renunciar a aprender algo. Depende de ti determinar qué quieres aprender y durante cuanto tiempo. Somos seres complejos y nuestros intereses se transforman y cambian. También estamos en una industria compleja de caprichos y tecnologías cambiantes. Lo que era interesante y necesario al comienzo del año, podría convertirse en algo carente de interés o innecesario al final del año. No debemos sentirnos presionados para aprender sobre algo solo porque otras personas lo estén aprendiendo o porque el mercado de trabajo parezca que lo requiera. Date permiso para escuchar tus propios deseos. Si estos coinciden con lo que quiere esta

industria voluble ¡entonces genial! Ve y aprende todo lo que puedas. Pero si no coinciden y te encuentras a ti mismo pasando semanas tratando de encontrar la motivación necesaria para aprender sobre el tema, entonces estás haciendo un flaco favor tanto a ti mismo como a tu oficio. Deja que el tema repose inactivo durante un tiempo y date tiempo para aprender sobre otra cosa. No tiene mucho sentido el sentirte miserable por complacer a otras personas.

Si sientes la necesidad de retomar el tema más adelante entonces permítete el regresar de nuevo a él. También deberías permitirte regresar a ese tema sin el bagaje y las expectativas de los intentos anteriores. El decir “ya intenté esto antes, así que veamos si funciona esta vez” pone a tu mente a la expectativa de que volver a abandonar de nuevo. Date permiso para acercarte a este tema como si estuvieras experimentando la experiencia por primera vez, sin expectativas sobre cómo resultará. Se amable contigo mismo y experimenta el tema de nuevo pero desde tu perspectiva actual.

6.8 Acércate con curiosidad

Cuando eramos principiantes nos acercamos a las computadoras con curiosidad y entusiasmo. No sabíamos qué esperar y no teníamos ni idea del tiempo que iba a llevar. Simplemente aprendimos tanto como pudimos y tomamos todo al pie de la letra. A medida que continuamos aprendiendo cambiamos nuestra curiosidad por certezas y nuestro entusiasmo por expectativas. La emoción que sentíamos al aprender se convirtió en un sentimiento de monotonía al sentir que siempre deberíamos estar aprendiendo. Podemos volver a capturar aquel espíritu del principiante viendo cada oportunidad de aprender como una nueva experiencia. Podemos dejar de lado nuestras expectativas sobre cómo progresará nuestro aprendizaje y en vez de eso acercarnos a cada sesión de aprendizaje con curiosidad por lo que aprenderemos durante la sesión. Podemos reavivar la llama que teníamos cuando eramos principiantes con infinitas posibilidades. Esa llama nos guiará cuando atravesemos periodos de incertidumbre.

Podemos volver a sentir pasión por aprender. Con cada contenedor de enfoque podemos acercarnos a nuestro aprendizaje de una manera fresca,

sin nociones preconcebidas de cómo y cuándo acabará, y ser curiosos por aquello que encontraremos cuando ahondemos más y más en lo que estamos aprendiendo. Cada sesión nos acerca un paso más en nuestro viaje por cerrar las lagunas de conocimiento. Hay mucho que explorar en nuestro campo. Espero que siempre encuentres algo nuevo y excitante que te ayude en tu viaje.

7 La lucha interna

7.1 Las emociones de la programación

Existe el estereotipo del programador sin emociones sentado frente a su equipo. El programador estereotipo se sienta, introduce líneas de código de manera silenciosa, como si las transcribiera desde su memoria. Si eres programador o has tenido contacto con programadores entonces deberías saber que el estereotipo debería ser el de un compositor frustrado. Por supuesto, nos sentamos delante de nuestros ordenadores en largos periodos en silencio y concentrados, pero estamos muy lejos de no tener emociones. Disfrutamos de las glorias de un código que funciona a la perfección la primera vez. Fruncimos el ceño cuando el código funciona mal. Pasamos de alegrarnos por la victoria a maldecir y amenazar a la máquina con los puños apretados. Apretamos nuestros dientes cuando los errores del código asoman. Pasamos de una emoción a otra: exuberancia, alegría, miedo, furia, resentimiento, tristeza, soledad, culpa o vergüenza.

No es extraño que al final del día estemos exhaustos.

Programar es un proceso agotador. No solo necesitamos mantener un modelo mental del software en el que estamos trabajando, también tenemos que mantener un modelo mental de cómo se debería comportar el software. Creamos una historia de cómo funcionará este software y componemos una imagen mental de cómo nos sentiremos cuando todo funcione como hemos lo hemos imaginado. Creamos un vínculo emocional con el software. Nuestro estado emocional puede reflejar lo que sentimos cuando estamos creando: excitados, aburridos o estancados. Mantener una actitud positiva respecto al software que no está a la altura de nuestras expectativas es agotador. Esto combinado con nuestras propias inseguridades, miedos y dudas nos da indicios cuando vemos que los programadores tienden a agotarse: es una combinación de estrés y nuestra reacción emocional a ese estrés.

7.2 Desgastes emocionales

Hay muchos factores que pueden causarnos altibajos emocionales mientras estamos programando. Estos son algunos que he anotado, tanto en mi propio proceso de programación como hablando con otras personas sobre su programación.

7.2.1 Propósito y utilidad

Si vemos con claridad dónde y cómo este código será útil, podemos tener una idea de la motivación y el propósito. ¡Estamos trabajando en algo que beneficiará a personas! Sabemos que las personas dependen de nosotros, así que hacemos todo lo que este en nuestra mano para hacer que el código funcione sin importar las trampas que nos esperen. Aprovechamos los picos emocionales de autoestima y determinación para ayudarnos a llegar a la culminación.

Lo opuesto también es cierto, por supuesto si no vemos el propósito entonces nuestro trabajo parecerá sin sentido y en vano. Nos esforzaremos para cumplir los plazos y sentiremos una sensación de vacío en nuestros objetivos. A veces se trata de un proyecto que no está en consonancia con nuestros propios propósitos y metas. Puede tratarse de un proyecto mal gestionado en el que estamos forzados a trabajar debido a presiones externas. Podríamos vernos obligados a cumplir plazos arbitrarios que nunca acordamos cumplir. Podemos sentirnos frustrados si no entendemos el fin último de cualquier proyecto en el que estemos trabajando.

7.2.2 Motivación frente a aburrimiento

Ya hemos experimentado diferentes capas de motivación con nuestra programación. Esos son los proyectos en los que no sentimos que sean una tarea mientras estamos trabajando en ellos. Sentimos que estamos aprendiendo algo en cada paso del camino. El mundo exterior desaparece mientras estamos trabajando inmersos por completo en ese enfoque.

Perdemos la noción del tiempo y nos sentimos tanto desorientados como renovados cuando el trabajo está completo.

Desafortunadamente lo más probable es que tengamos más experiencias con lo opuesto a la motivación: el aburrimiento. El código base no nos motiva en absoluto. El tema que estamos aprendiendo o en el que estamos trabajando es simplemente una repetición de algo que ya sabíamos. Es un suplicio comenzar. Todo lo que nos rodea en el mundo parece más interesante y el tiempo parece ir más despacio durante todo el proceso.

7.2.3 Despierto frente a cansado

El sueño es un factor que influye mucho en cómo percibimos el mundo. Dormir lo suficiente nos permite sentirnos renovados, despiertos e inspirados. Necesitamos tener reservas de energía para enfrentarnos a cualquier reto que se nos presente. Cuando no tenemos una calidad óptima del sueño nos volvemos irritables y menos dispuestos a la participación. Conservamos nuestros recursos lo mejor que podemos para que no se gasten demasiado rápido. Buscamos estimulantes (café, distracciones y similares) para mantenernos ocupados a lo largo del día.

7.2.4 Estado mental

Uso el término “estado mental” en un sentido amplio para cubrir cualquiera de nuestros sentimientos existentes y de bienestar mental actual. Estos pueden variar desde sentimientos temporales de infelicidad o melancolía a sentimientos complejos y serios como la depresión clínica o trastorno de estrés postraumático. Nuestras mentes son máquinas complejas que hacen lo que pueden para adaptarse a las diferentes situaciones y entornos que se les presentan. En ocasiones esta adaptación puede chocar con nuestros deseos de ser productivos y la lucha que se produce entre nuestro estado mental y nuestros deseos puede causar más desgaste emocional, incomodidad o desesperación.

Hay más motivos que pueden afectar a nuestras emociones pero estos son en los que me gustaría centrarme ya que cubren un amplio espectro de lo

que acarrea dedicarnos a las tareas de aprendizaje y programación.

7.3 Ser conscientes de nuestro estado emocional

Ser conscientes de nuestro estado emocional (lo que estamos sintiendo ahora mismo) nos da nuestra ubicación emocional actual. Podemos ubicarnos dónde estamos y entender lo que nuestra mente nos está diciendo. El darnos unos instantes para sentir verdaderamente en qué estado emocional se encuentra nuestra mente nos ayudará a seguir adelante.

Ten en cuenta que no estamos tratando de cambiar nuestro estado emocional. No nos estamos intentando forzar a ser algo que no somos. Si realmente estamos insatisfechos sobre dónde estamos o lo que estamos haciendo, es más útil entender qué está causando esa infelicidad en vez de tratar de disimular y prevenir esas emociones. Observar nuestras emociones de manera clara nos permite reconocer qué las está causando. Estar presente con esas emociones nos permite entender mejor nuestro estado mental y de lo que somos capaces en el momento.

Puedes realizar esto dentro de un contexto de meditación de atención plena, incluso mientras estás sentado en tu escritorio pensar: “durante uno o dos minutos voy a estar sentado aquí y voy a explorar mi estado emocional” debería ser suficiente. Ser conscientes de nuestras emociones, entender qué son y profundizar en ellas hasta encontrar qué las está causando puede ayudarnos a entender qué estamos sintiendo.

Quizás ya sabes qué está causando esas emociones y ese estado mental y tienes miedo de explorarlas. Algunas emociones pueden sobrepasarnos y hacernos sentir de formas que no queremos sentirnos. Esto es especialmente cierto con emociones relacionadas con la ansiedad y el trastorno de estrés postraumático. Realiza toda la exploración e introspección que seas capaz, y se amable contigo mismo. Recuerda, no estás tratando de cambiar las emociones, solo estás siendo consciente de ellas. Es posible que descubras que tu insistencia amable en estas emociones pueda llevarte a entenderlas mejor. Se tan osado como puedas con estas emociones y si empiezan a

abrumarte, entonces retrocede y deja que desaparezcan los residuos de esas emociones antes de continuar.

7.4 Nuestra historia

Cada uno de nosotros tenemos una historia que nos contamos a nosotros mismos. Estas historias conforman nuestra percepción del mundo. Nos contamos historias de cómo transcurrirá el día y cómo nos enfrentaremos a el. Creamos un mundo a través de nuestras historias en el que somos el protagonista central de la historia. Nos contamos historias como “el trabajo que voy a comenzar será increíble” o “voy a trabajar en este problema y cuando termine obtendré una solución genial”. Eso ocurre si estamos siendo positivos con nosotros mismos. Cuando estamos siendo negativos, nuestras historias tratan sobre que no somos lo suficientemente buenos en lo que estamos haciendo y que seguramente fallarás en el intento. Estas historias crean una trama compleja de lucha, dolor y miseria donde todo lo malo del mundo es el resultado directo de nuestras acciones.

Nuestras emociones ayudan a dar forma al tipo de historia que contamos. Si nos sentimos muy bien, nos diremos a nosotros mismos que lo que está por venir también será genial. Si nos sentimos deprimidos y derrotados, nuestras historias reflejarán ese tono derrotado.

La verdad es que nuestra historia es simplemente eso: una historia. Nuestras historias no son una garantía de cómo va a transcurrir el día. Podemos contarnos una historia sobre que hoy será un día increíble y ver con horror que cada interacción hace que nuestro día sea de todo menos increíble. Por el contrario, nuestra historia podría ser que hoy será un día terrible y que no conseguiremos nada, pero en vez de eso experimentamos un día muy completo y productivo. La historia solo acentúa lo que estamos experimentando, pero no puede predecir lo que experimentaremos.

En vez de apegarnos a estas grandes historias podemos centrarnos más en las cosas que amamos del momento presente. En vez de una historia en la que vas a tener un día excepcional podrías centrarte en aspectos de tu proyecto que te atraigan y esperar en que puedas pronto trabajar en ellos.

En vez de llenar tu día con historias de terror y fatalidad te puedes centrar en las pequeñas victorias que ocurren a lo largo del día. Incluso algo tan simple como “mi ordenador ha arrancado sin fallar” puede ser una victoria. Una de esas pequeñas victorias podría ser el establecer una intención en permanecer centrados y curiosos en los próximos 10 minutos (el contenedor de enfoque de los capítulos anteriores) y celebrarlos cuando logres esa intención. Puedes tener más pequeñas victorias si te mantienes trabajando con esa intencionalidad a lo largo del día. Nuestras pequeñas victorias no serán perfectas (quizás tu ordenador está hoy insoportable), pero podemos usarlas para recalibrar nuestro día en los próximos 10 minutos y seguir usándolas para recalibrar durante todo el día, como cada contenedor focalizado se convierte en otra pequeña victoria.

Darnos a nosotros mismos la capacidad de centrarnos más en el presente y los próximos pasos que estamos a punto de dar nos brinda una forma consciente de comprobarnos a nosotros mismos y nuestro progreso. Podemos centrarnos en los aspectos positivos de lo que estamos haciendo en vez de preocuparnos sobre cómo la realidad difiere de nuestras propias historias. Podemos corregir el rumbo a lo largo del día y mantener la tendencia hacia un día más positivo y productivo en vez de preocuparnos sobre lo distantes que nos encontramos de nuestro día ideal.

Esto requerirá práctica. Estamos acostumbrados a permitir que nuestras historias dirijan nuestro día, pero con el paso del tiempo seremos capaces de dividir nuestro días en partes más pequeñas donde podamos ser plenamente conscientes de las historias que nos contamos a nosotros mismos.

7.5 La conciencia en acción

Vamos a suponer por un momento que es un día normal para nosotros. Hoy nos sentimos ansiosos. Hemos recibido un informe de un error que está relacionado con algo en lo que hemos estado trabajando. En informe de error indica que el código que enviamos al proyecto a principios de año no está funcionando y quizás nunca funcionó de la manera que pensamos que funcionaba. Mientras leemos el informe de error nuestros niveles de ansiedad se incrementan. Nuestro monólogo interior entra en acción y

empezamos a decirnos a nosotros mismos que nunca hemos estado cerca de ser tan buenos como creíamos. No somos perfectos. Apestamos. No hemos dormido la suficiente la noche anterior así que nuestras emociones están a flor de piel. Nuestra mente se acelera y proyecta imágenes pasadas de otras veces donde también cometimos fallos. Mientras seguimos leyendo aflora nuestra sensación de temor. Nuestro monólogo interno se convierte en una charla frenética: "¿Qué van a pensar ahora de mí? ¿Qué piensan de mí ahora mismo? ¿Perderé mi trabajo por esto?

Antes de que hayamos acabado de leer el informe de error ya hemos creado una historia. La historia comienza con nuestra propia ansiedad por lo que pasará a lo largo del día. Entonces ocurre lo peor: obtenemos algo que confirma nuestros miedos. La historias después nos presenta con un montaje de nuestros fallos pasados y a eso añade este último informe de error como punto culminante del montaje. Nuestra historia entonces aumenta la presión por el aumento de la importancia de este informe de error: no solo debemos solucionar lo que sea que esté roto, ahora tendremos que limpiar nuestra reputación y comenzar a buscar trabajo. A medida que la historia progresa en nuestras mentes, nos preguntamos si volveremos a trabajar como programador de nuevo y sentimos cómo nuestra carrera profesional como programador está acabada.

La historia que hemos creado es una historia terrible, pero estoy seguro que te ves identificado con los factores que genera. Es una historia que se basa en lo más profundo de nuestros propios sentimientos de insuficiencia e inseguridad. Está alimentado por el miedo: el miedo a arruinar tu reputación, miedo a que no confíen en ti y miedo al fracaso.

El miedo es una de las emociones más poderosas que tenemos, pero no es la única. Leer ese informe de error también puede hacer surgir otras emociones como el dolor (pensamos que el código era bueno y ahora ese sentimiento se ha esfumado). Incertidumbre (¿cómo vamos a solucionar el problema?) y la ira (¿cómo hemos podido habernos engañado pesando que esto funcionaba?) También podemos tener otros sentimientos: tristeza, soledad y abandono. Nuestra autoestima podría verse afectada y podríamos sentirnos desconectados de aquellas personas para las que trabajamos y las personas con las que trabajamos.

Estar al tanto de estos sentimientos puede ayudarnos a analizar la historia que nos estamos contando y cómo no coincidía con la realidad. Estos sentimientos y la historia que nos contamos nos dan una retroalimentación de cómo percibimos el mundo y el mundo que estamos construyendo. Hacer una pausa por un momento para reconocer nuestros sentimientos y entender de donde proceden nos da un entendimiento de lo que nuestras emociones están tratando de decirnos.

Ahora te puedes relajar. El informe de error de este libro no es real, pero tómate un momento para reconocer los sentimientos que sientes cuando lees la sección anterior y nota dónde ha ido tu mente. Este es el tipo de conciencia que estamos buscando tener.

7.6 Hallar nuestros sentimientos

Nuestros sentimientos se manifiestan en nuestros cuerpos de muchas maneras diferentes. El miedo puede sentirse como un nudo en el estómago o una tensión en nuestro pecho. La ira puede hacer que apretemos las mandíbulas o sentir nuestra cabeza más caliente de lo normal. La tristeza la podemos sentir como un peso sobre nuestros hombros o hacernos sentir cansados. Cuando notamos estos sentimientos podemos hacer una pausa por un momento y simplemente sentarnos con nuestros sentimientos mientras lo seguimos sintiendo.

Piensa en este ejercicio como si estuvieras escaneando tu cuerpo en busca de la fuente de los sentimientos que estás sintiendo. Fíjate hacia donde es atraída tu mente: opresión en el pecho, una opresión en tu estómago, las mandíbulas apretadas o cualquier otro sentimiento que estés sintiendo. Nota la sensación de ese sentimiento. Puedes ahondar más y tratar de encontrar la causa primigenia del sentimiento pero por ahora simplemente siente que existe. Siéntate por un momento y se más curioso sobre cómo te sientes. Trata de notar otros atributos de ese sentimiento: color, textura, intensidad o cualquier otro atributo que estés experimentando. Deja que ese sentimiento exista, se amable con el. Permite que exista sin juzgarlo. Dale espacio. Sobre todo no trates de luchar contra el sentimiento o desees que termine, simplemente siéntelo. De manera eventual puede que el sentimiento vaya

disminuyendo, pero por ahora solo ten conciencia de que tienes ese sentimiento y que vas a ser curioso sobre él.

Algunos sentimientos y emociones son más dolorosas o traumáticas que otras. Dales espacio y permítete tener curiosidad sobre ellas mientras puedas. Si notas que tu mente empieza a sentir pánico u otra sensación que te desborde debido a este sentimiento, entonces deberías dejar de notarlos antes de que te superen. Recuerda que esto son emociones y que esas emociones son parte de ti. Tanto tu como tus emociones trabajáis juntos para ayudarte. Ambos estáis en el mismo equipo.

Con este ejercicio no se trata de que te acoses o te castigues con tus propios sentimientos. Es solo el acto de notar que estos sentimientos te causan un dolor físico o emocional, es posible que necesites ayuda profesional o un grupo de apoyo para guiarte a entender estos sentimientos y descubrir de dónde vienen. La ayuda profesional o el grupo de apoyo pueden ayudarte a tener estos sentimientos sin que estos te sobrepasen. No debe provocarte vergüenza el buscar ayuda en otras personas para que te ayuden en tu viaje.

7.7 Separación y selección emocional

Uno de nuestros comportamientos aprendidos respecto de nuestros sentimientos es tratar de huir de ellos o de reprimirlos. Hacemos todo lo posible para evitar sentimientos que nos provocan infelicidad o incomodidad. También tratamos de no exagerar nuestros sentimientos positivos para no mostrar demasiada exuberancia. Esto nos puede llevar a sentirnos confusos o en conflicto sobre qué estamos sintiendo y por qué nos sentimos de esa manera. Al sentarnos con nuestros sentimientos y emociones y entendiendo de dónde proceden podemos formarnos una idea clara de qué está pensando nuestra mente y la historia que nos estamos contando a nosotros mismos.

Piensa en esta práctica como una separación y clasificación emocional. Con suerte nunca has tenido que acudir a una sala de emergencias de un hospital, pero si lo has hecho, verías a todo el personal médico que están entrenados en diagnosticar a quien acaba de entrar por la puerta y determinar la

gravedad del problema. Cuando reconocemos y reflexionamos sobre nuestras emociones también estamos diagnosticando qué emociones estamos teniendo y la gravedad de esas emociones. Tomamos esos momentos cuando estamos experimentando las emociones para poder determinar qué emociones son y qué las ha desencadenado. Mientras revisamos nuestras emociones somos cordiales con ellas y las reconocemos por lo que son. Un buen profesional médico no impone sus propios deseos al paciente, simplemente aceptan a los pacientes por lo que son, diagnostica lo que está experimentando el paciente y actúa en consecuencia. Cuando reconocemos nuestras emociones por lo que son y determinamos de dónde proceden, podemos comprender mejor a lo que nos estamos enfrentando.

Cuanto más hagamos esta práctica, mucho mejor seremos capaces de reconocer nuestras emociones y por qué las estamos teniendo. Seremos más capaces de notar qué estamos sintiendo y entender el motivo de estar sintiéndonos de esa manera. Cuando sentimos ansiedad, podemos reconocer que podría ser causada por explorar un área de programación que no entendemos por completo. Podemos sentir esa ansiedad durante un instante (no trates de ahuyentarla) y en ese momento piensa sobre lo que estás trabajando y cómo poder explorar esas áreas que son nuevas. Podemos tomar una nota mental o escrita (mejor si es en un diario) así cuando completemos lo que estamos realizando, podremos revisar las áreas que nos están causando ansiedad.

Esta práctica puede convertir nuestras emociones en algo que nos dirija o guíe. Podemos utilizar nuestras emociones como herramientas para calibrar mejor nuestras historias internas. Podemos recomponer esas historias sobre cómo no nos merecemos que nos llamen programadores y en vez de eso darnos la intención de que pasaremos los próximos 10 minutos explorando este área de nuestro trabajo y encontrar dónde están las lagunas de conocimiento. Podemos establecer una intención de ser curiosos sobre a dónde nos llevarán los próximos 10 minutos de exploración. Mientras continuamos explorando ese tema notaremos nuestras emociones y usaremos esas emociones para permitirnos conocer dónde necesitamos mejorar y adaptarnos. Esto nos permitirá cambiar nuestros planes cuando sea necesario y abordar aquellas áreas donde creamos que tenemos carencias o necesitan mejorar. Este ciclo continúa en cada contenedor que

practiquemos, con nuestras emociones actuando como un barómetro de nuestro nivel de comodidad con el tema en cuestión y ayudándonos a esbozar una hoja de ruta sobre la mejor manera de proceder. Transformamos nuestra incomodidad y ansiedad de cosas que obstaculizan nuestro progreso en indicadores de dónde sentimos que tenemos que enfocar nuestra atención.

7.8 Agotamiento

Una de las cosas que nuestra separación emocional puede ayudarnos a diagnosticar es el sentimiento de agotamiento. El agotamiento es un compendio de emociones unidas a una extenuación emocional y física. El agotamiento puede ser algo simple como estar cansado o tener exceso de trabajo, pero también puede ser un signo de algo más serio. Puede llevar a complicaciones físicas o mentales si no tenemos cuidado. Podemos trabajar con preocupantes niveles de extenuación y engañarnos a nosotros mismos creyendo que esto es parte del coste por ser un buen programador.

El agotamiento se manifiesta de diferentes maneras. Para algunas personas puede ser un sentimiento de pavor mientras trabaja en un proyecto. Se sienten como si no fueran efectivos al hacer cualquier cambio. Para otras personas el agotamiento puede ser un sentimiento de extenuación. Se sienten como si estuvieran en una rueda de hamster que no parará nunca. Incluso puede ser mucho peor, quieren que la rueda se hubiera parado hace ya mucho tiempo. El agotamiento también puede manifestarse en una sequía creativa, donde imaginar un futuro diferente es difícil y las cosas que una vez fueron inspiradoras o interesantes ya hace tiempo que no generan esa chispa.

El agotamiento es complicado de auto diagnosticar debido a que es un conjunto de emociones aparentemente no relacionadas entre sí. Nuestro sentimientos de aburrimiento, miedo y ansiedad pueden tener todos una causa raíz diferente, pero cuando combinamos estos sentimientos con un horario de trabajo implacable y la pérdida de control, amplificamos esos sentimientos. Si no los controlamos, nos pueden llevar a tratar de silenciar o dormir esos sentimientos. Nos encontraremos con que no queremos

programar nunca más y nos enfadaremos con nosotros mismos por habernos metido en la programación. Podemos causarnos más sufrimiento no deseado simplemente “superándolo”, lo que puede llevarnos a agravar y complicar aún más nuestro estado emocional.

Hay algunas cosas que podemos hacer para entender y ayudar a aliviar ese agotamiento:

- Darse cuenta de cuando estamos agotados, o a punto de estarlo. Reconocer que estamos a punto de sentirnos agotados es clave para no llegar a experimentar el agotamiento. Esto parece bastante simple, pero tendemos a ignorar los síntomas cuando nos estamos acercando al agotamiento. Si podemos reconocer que estamos a punto de agotarnos entonces podremos tomar medidas para evitarlo. Y si nos damos cuenta que ya estamos agotados, podemos tomar medidas para ser amables con nosotros mismos y ayudarnos a salir de este estado de agotamiento.
- Examinar nuestras emociones. Siéntate un momento y observa qué emociones aparecen delante de ti. ¿Estás sintiendo estrés, miedo, ansiedad, nerviosismo o ira? Observa qué sentimientos emergen y reconoce esos sentimientos. Examina de dónde provienen esos sentimientos y qué podría estar provocando esas emociones.
- Volver a negociar nuestros compromisos. Muchas veces el agotamiento es el resultado de un compromiso excesivo, ya sea con nosotros mismos o con otras personas. Siempre tenemos muchas cosas que hacer y a pesar de nuestros mayores esfuerzos siempre adquirimos nuevas obligaciones. Quizás los planes que hicimos fueron muy exigentes o algo ha cambiado en el mundo que desbarató nuestros planes. Cualquiera que sean las razones, necesitamos volver a evaluar lo que se esperaba de nosotros y lo que somos capaces de hacer. Si vemos que hemos creado una situación intratable para nosotros mismos, necesitamos averiguar cómo eliminar parte de esas obligaciones o volver a negociarlas.
- Dar un descanso a nuestro ímpetu. A diferencia de nuestros homólogos mecánicos necesitamos tiempo de inactividad y descanso. No podemos trabajar ocho o más horas sin al menos algunos momentos donde no estemos trabajando. La programación requiere mucho ancho de banda

mental y el esforzarnos hasta la extenuación puede llevarnos a inestabilidad, estrés y agotamiento.
- Examinar si así es como realmente queremos vivir nuestras vidas. Necesitamos determinar si lo que estamos haciendo es realmente lo que queremos hacer. Si no somos felices con lo que estamos haciendo, entonces cada momento que continuemos haciéndolo puede agravar nuestros sentimientos de infelicidad. Si no sentimos más que terror por nuestra situación actual entonces quizás necesitamos volver a negociar nuestros compromisos. Eso puede ser algo simple como aceptar no aprender algo ahora mismo, o puede ser algo complejo como cambiar de trabajo o cambiar de carrera.

Al entender que estamos dirigiéndonos hacia el agotamiento (o que ya estamos en él) podemos tomar medidas para corregir el rumbo, para poder así realizar nuestra programación disfrutando y con alegría. A veces dar un paso atrás y volver a evaluar lo que estamos haciendo nos puede ayudar a no instalarnos en bucles constantes de frustración, ira y culpa. Cambiar nuestra historia para que se ajuste mejor a la realidad puede evitar que intentemos alcanzar un sueño imposible.

Antes mencioné la posibilidad de volver a negociar los compromisos. A menudo estamos involucrados en situaciones donde tenemos muchas más tareas que hacer que lo que es físicamente posible, incluso bajo las circunstancias más óptimas. Esto puede ser causado en parte porque hemos dicho “sí” a demasiadas cosas, o porque estamos abrumados por compromisos laborales, como un importante proyecto de alta prioridad o varios proyectos más pequeños que necesitan atención urgente. La mejor manera de volver a negociar tu carga de trabajo es revisando esa carga de trabajo y anotar qué tareas son “urgentes” y qué tareas son “importantes”. Las tareas “urgentes” son las que sientes que hay que realizarlas de manera inmediata. Puede que no sean tareas “importantes”, pero tienen un sentido de urgencia. Las tareas “importantes” son las que de alguna forma te beneficiarán a ti o a otras personas. Estas tareas tienen un valor significativo cuando se finalizan, ya sea monetario o por lo que significa acabar dichas tareas. Toma una hoja de papel o abre un documento de texto y crea dos categorías: “urgente” e “importante”. Haz una lista con las tareas que debes completar y clasifícalas bajo una de las dos categorías. A continuación

marca la fecha de finalización (lo más aproximada que puedas) de cada una de estas tareas. Si tienes más de tres tareas urgentes o importantes y todas ellas finalizan en la misma semana es probable que tenga exceso de trabajo y necesitarás volver a negociar esos compromisos. Puede que te sientas capaz de hacer todas esas cosas pero si ya te sientes estresado, cansado y agotado, al tratar de cumplir esos plazos solo aumentarás esos sentimientos. Si es posible, trata de mover algunos de esos plazos para la próxima semana o comprueba si tus clientes creen que esas tareas son tan urgentes o importantes como tu creías que eran. Si realmente son urgentes o importantes, entonces trata de que tus superiores te ayuden con recursos o para ver si esas personas pueden intervenir para volver a negociar los plazos y las prioridades. Si estás atascado sin salida (tus responsables no va a interceder y los clientes no van a volver a negociar los compromisos) entonces tendrás que tomar algunas decisiones sobre cómo son de importantes esas prioridades frente a tus propias prioridades (esas tareas ayudan a tus ingresos, lo que contribuye a continuar con tu estilo de vida), pero tu propia salud y tu propio bienestar debería tener más peso en la toma de tu decisión que sus prioridades o sus plazos de entrega. Quizás puedas negociar algo de tiempo libre o vacaciones después de ese periodo para poder descansar, relajarte y recuperar tu fortaleza y agudeza mental antes de volver a pasar por una situación parecida.

Aprender a decir “no” es una capacidad muy importante como programador. A menudo nos vemos a nosotros mismos como seres extraordinarios que pueden hacer cualquier cosa que se propongan, en parte porque los equipos informáticos con los que trabajamos parece que pueden hacer cualquier cosa. Desafortunadamente, tenemos unos recursos físicos y emocionales finitos, así que aprender a escoger y elegir los proyectos que son más importantes para nosotros (dependiendo de nuestro propio criterio) nos ayudará a mantenernos a medida que vamos avanzando en nuestra carrera profesional como programadores. Si decimos que “sí” a todo lo que alguien escoge por nosotros entonces tendremos menos tiempo para trabajar en las cosas que realmente nos importan. Estaremos a merced de personas cuyas prioridades y deseos no coinciden con los nuestros. La forma más efectiva de agotarse es gastar toda tu energía trabajando en proyectos que no encajan con tus prioridades y deseos.

Experimentarás periodos de agotamiento en tu carrera profesional como programador. Te encontrarás con cosas que sobrepasarán tu capacidad para hacerles frente. Te verás atrapado en bucles en los que te preguntarás si esto es realmente lo que deberías estar haciendo. Comprender lo que estás sintiendo y reconocer que esos sentimientos son válidos es el primer paso para cambiar tu trayectoria del agotamiento y el estrés. La programación no debería ser una tarea tediosa (ningún trabajo de valor debería serlo). Debería haber algo en tu día a día como programador que te mantuviera motivado y te ayudara a aumentar tus conocimientos. Añadir pizcas de conocimiento, disfrute y asombro, junto con periodos de inactividad, te ayudará a mantenerte a salvo en la turbulencia emocional que te espera. Y reconocer cuando estás agotado y volver a negociar tus compromisos contigo mismo y con otras personas puede ayudarte a revitalizar tus deseos de continuar programando.

7.9 Buscar ayuda

Me gustaría enfatizar el hecho de que está bien el pedir ayuda a otras personas. Personalmente me ha costado pedir ayuda. Parte de mi reticencia a la hora de pedir ayuda estaba provocada por que cada vez que hacía una pregunta obtenía la temida respuesta de “eso ya deberías saberlo”. Otras veces creía que al pedir ayuda, eso de alguna manera haría mermar mi reputación. Me considerarían un fraude o un impostor. Las personas se preguntarían por qué habrían confiado en mí al principio. Pero cuando finalmente pedí ayuda, las respuestas no fueron “¿por qué no sabes sobre eso?” si no que fueron “¿por qué no has pedido ayuda antes?” Claro que hubo ocasiones donde alguien se habría sorprendido de que no supiera algo, o que recibiría críticas por mi ignorancia, pero encontré que los beneficios de preguntar superan con creces cualquier efecto negativo.

Pedir ayuda no está limitado únicamente a temas técnicos, hay muchas otras maneras en las que podríamos necesitar ayuda. Podríamos preguntar a colegas que nos ayudasen durante un periodo difícil de nuestras vidas. Podríamos necesitar ayuda de nuestros superiores cuando estamos luchando de alguna manera ya sea personal o profesionalmente. Podríamos incluso necesitar diferentes tipos de ayuda de diferentes profesionales (doctores,

terapeutas, etc). Involucrar a otras personas en nuestros problemas puede ser desalentador (o incluso abrumador) pero obtener ayuda de manera temprana puede ayudarnos a prevenir formas más graves de agotamiento o estrés.

La razón más común de nuestra reticencia a la hora de pedir ayuda es nuestro deseo de confort. Pedir ayuda significa el ponernos a nosotros mismos en un estado de vulnerabilidad y esperar que las personas a las que pedimos ayuda nos traten con amabilidad, respeto y dignidad. Esta vulnerabilidad puede verse amplificada si no conocemos a la persona a la que pedimos ayuda o si esa persona es un profesional médico. Pero es necesario el colocarnos en esa situación de vulnerabilidad, especialmente si los problemas o situaciones a las que nos enfrentamos están fuera de nuestro control o exceden nuestra experiencia. Si estamos a punto de sentirnos agotados (o estamos inmersos en un proceso de agotamiento) podremos necesitar la ayuda de un doctor o terapeuta para tratar de la mejor manera la situación que estamos experimentando. Si nuestro trabajo nos causa estrés o tensión, quizás necesitemos hablar con otras personas de nuestro entorno para descubrir si también otras personas están experimentando estos sentimientos. Incluso un simple acto de compasión con nuestros compañeros puede ayudarnos a darnos cuenta que no estamos solos enfrentándonos a estos problemas y puede ayudarnos a encontrar mejores formas de gestionar nuestra carga de trabajo o estrés. A veces no nos damos cuenta de cuando nuestros trabajos o relaciones han pasado a ser de respetuosas y beneficiosas a ser algo que nos provoca más daños que algo positivo.

“No hay que sentir vergüenza en pedir ayuda” es una frase demasiado utilizada, pero pedir ayuda no es un acto vergonzoso. Necesitamos ayuda de otras personas. Incluso simplemente alguien que diga “siento que tengas que tratar con eso” puede ser una conexión con otra persona que simpatiza con lo que estamos pasando. Encontrar a otras personas que estén dispuestas a escuchar, empatizar y compadecerse puede ser la diferencia entre sentirse parte de una comunidad o sentir que hemos sido abandonados en nuestra profesión.

También necesitamos reconocer cuando nuestros sistemas de apoyo no nos están siendo de ayuda. Si vemos que hablar con alguien más no nos está ayudando a resolver los problemas, quizás necesitamos encontrar otras vías de ayuda. Quizás nos demos cuenta que necesitamos ayuda extra.

Darse cuenta de la necesidad de apoyo adicional puede ser difícil, pero una vez que hayas llegado a la conclusión, te animo a que actúes para conseguir esa ayuda extra. Esto requiere de auto conciencia y honestidad con lo que estás sintiendo. Solo tu conoces tu situación y si estás siendo honesto contigo mismo. Si no estás siendo honesto contigo, entonces solo tu puedes darte cuenta de eso y puedes tomar la iniciativa para buscar la ayuda que necesites. Nadie mejor que tu conoce tus propios mecanismos internos.

Pedir y recibir ayuda es una habilidad y como cualquier otra habilidad requiere práctica. Cuando éramos niños, teníamos medios muy simples para pedir ayuda (llorar, señalar algo, etc). Estas habilidades están grabadas en nosotros como parte de nuestros mecanismos de supervivencia, pero a medida que crecemos nuestro mundo se vuelve más complejo. Nuestros métodos de pedir ayuda tienen que madurar igual que nosotros mismos vamos madurando. Esto no es algo que surja de manera natural en ninguno de nosotros. Nos resistiremos a pedir ayuda y nos resistiremos cuando estemos recibiendo ayuda de otras personas. La repetición y una práctica cuidadosa nos ayudará a mejorar nuestra habilidad de pedir ayuda. Mejorar esas habilidades nos ayudará a sobreponernos a los obstáculos a los que nos enfrentamos en nuestro día a día. Esa mejora nos ayudará a ser no solo mejores programadores si no también a mejorar cómo gestionamos los retos que la vida nos ofrece.

7.10 Renunciar

A los programadores no les gusta pensar en darse por vencidos. ¿Cuantas veces hemos pedido a otras personas que tengan paciencia mientras intentábamos solucionar algo que no estaba funcionando? (“Déjame unos minutos más, por favor, ¡de verdad!”) Trabajamos con máquinas que parece que no tienen límites en sus posibilidades. Como programadores nos sentimos obligados a explorar esas posibilidades, pero a veces no queremos

hacer esa exploración. A veces echamos un vistazo a la lista de cosas que deberíamos aprender y nos preguntamos si merece la pena el esfuerzo. Miramos las ofertas de empleo relacionadas con nuestras habilidades y no encontramos mas que trabajos sin sentido. Los nuevos programadores nos preguntan qué es ser un programador y nos preguntamos si deberíamos advertirles sobre los peligros de escoger una profesión que nos ha llevado a ser infelices y sentirnos insatisfechos. La alegría que sentíamos mientras aprendíamos el oficio va desapareciendo y luchamos contra el miedo de nunca volver a sentir ese sentimiento de nuevo.

La programación no es para todo el mundo. Ha habido momentos donde me he preguntado si debería continuar trabajando como programador. Estoy frustrado porque no puedo aprender todo lo que quiero saber. Me preocupa que lo que estoy aprendiendo siga siendo relevante cuando haya acabado. Siento ansiedad por no se capaz de competir en un mercado laboral donde todos los demás parecen tener ventaja. Lucho por conseguir un puesto de trabajo que ofrece un puesto que no creo que vaya a ser relevante dentro de seis meses, y ya no digamos 10 o 100 años. Siento que el futuro de la computación que me prometieron ha sido corrompido y todos nosotros estamos atrapados en un mundo donde los ordenadores no son más que simples palancas para que las empresas abran las carteras de sus clientes.

Es fácil volverse fatalista sobre la práctica de la programación, pero me he dado cuenta de que hay más en la informática y la programación que lo que el mercado laboral ofrece.

Parte del placer de programar es la curiosidad. Si podemos alimentar nuestra curiosidad mientras programamos, entonces tenemos muchas vías por explorar. Siempre hay otras ideas y otros temas por descubrir, como el desarrollo de juegos, lenguajes de programación “esotéricos” u otros paradigmas de programación. El mercado laboral utiliza una fracción de las ideas de programación que están esperando a ser exploradas. También hay muchos emuladores y equipos antiguos disponibles con buena documentación y comunidades vibrantes. Un área que me ha intrigado, es aprender cómo funcionan las computadoras más antiguas. Las computadoras más antiguas son simples y se pueden entender con paciencia y la mentalidad adecuada. Estas máquinas son bien conocidas y la mayoría

de estos programas más antiguos fueron elaborados por un solo programador. Constituyen excelente espacio para aprender no solo cómo funcionaban las máquinas más antiguas, sino también muchos de los conceptos que aún impregnan nuestras máquinas modernas.

¿Qué ocurre cuando nos damos cuenta de que no quedan en nosotros ganas de disfrutar al programar? ¿Qué hacemos cuando la idea de programar ya no nos emociona? ¿Cómo continuamos cuando la simple idea de probar algo nuevo nos llena de terror? ¿Entonces qué?

Si no encontramos ninguna diversión en programar, entonces necesitamos entender por qué nos estamos sintiendo de esta manera. Quizás estamos cansados después de un proyecto muy duro que ha agotado en nosotros toda la diversión y la emoción de programar. Quizás las comunidades a las que nos hemos unido, ya sea en nuestra ciudad o por la red son hostiles o poco acogedoras. Quizás pensemos que programar sería más divertido pero cada vez que empezamos, desearíamos estar haciendo *algo* / *cualquier cosa* en su lugar.

Programar es lo mejor cuando de verdad quieres hacerlo. No es para cualquiera. Si estás estancado en una situación en la que no quieres programar nunca más, entonces el mejor rumbo que puedes tomar es dejar de ser programador. No tiene que ser vergonzoso dejar de programar, muchos programadores han perdido la chispa de la ilusión y el deseo de programar y se han cambiado a otros campos. Está bien dejar el campo de la programación y hacer otra cosa.

La programación solo es una faceta de nuestras vidas. Cierto, puede ser una faceta muy grande de nuestras vidas y puede dar miedo el abandonar algo en lo que hemos trabajado tan duro para conseguir. Pero si nos damos cuenta que únicamente seguimos los movimientos por inercia y ya no experimentamos ningún disfrute cuando estamos programando, entonces es tiempo de pensar sobre qué más podemos hacer con nuestras vidas fuera de la programación. Se nos concede una cantidad limitada de tiempo para vivir nuestras vidas. Hacer algo con lo que no disfrutamos nos impide vivir una vida más significativa.

Renunciar no debe ser una experiencia negativa. No hay vergüenza en quitarle tiempo a ser programador. Muchos programadores se han dado un “año sabático” de la programación para explorar otros intereses y recargarse de energía. Romper bucles de experiencias negativas en la programación puede ayudarnos a identificar lo que queremos de la programación y de la profesión de programador. Puede ayudarnos a encontrar y confirmar nuestros sentimientos más íntimos sobre la programación y ver si todavía estamos destinados a seguir por este camino.

Hay varios miedos que pueden impedirnos hacer esta ruptura con la programación. El primer temor se conoce con el elegante nombre de “falacia de la inversión irrecuperable”. La falacia de la inversión irrecuperable es la creencia de que el tiempo y el esfuerzo que dedicamos a aprender y programar es una inversión, y esa inversión se desperdiciará si renunciamos. Por lo tanto, para preservar nuestra inversión debemos seguir programando. El problema con esta falacia es que supone que aún no hemos recibido el beneficio de ese tiempo y esfuerzo. Yo diría que aprender cualquier tipo de programación no es una habilidad desperdiciada. La programación se puede aplicar en muchas facetas de nuestras vidas, como simplificar tareas dividiéndolas en porciones más manejables, aplicar el pensamiento estructurado y comprender la lógica booleana básica. Muchos otros campos también han adoptado las computadoras, por lo que tener habilidades informáticas puede ser útil para ti o para otras personas. El conocimiento que tienes no se desperdiciará.

El segundo miedo es el miedo a defraudar a nuestros compañeros programadores y a otras personas de nuestra organización si abandonamos la programación. Esto es complicado. Es complicado porque incluye a otras personas en nuestro proceso de toma de decisiones. Es posible que estemos en una organización que tiene una carga sustancial de tareas para completar, y nuestra decisión de renunciar significará que estas tareas no se completarán de la manera en que deseamos que se completen. No es difícil imaginar que nuestra ausencia cause daño a toda la organización y resulte en su eventual colapso. ¿Es cierto este escenario? Depende de nosotros determinar si nuestra ausencia realmente decepcionará a todos en nuestra organización. Esto nos pone en una situación en la que nuestro miedo nos mantiene sintiéndonos “acorralados”. Nos sentimos “acorralados” porque

nuestro miedo ha creado una situación en la que estamos eligiendo entre nuestro propio bienestar o el bienestar de los demás. Esta es una falsa dicotomía. Nuestra ausencia podría ser el catalizador para que alguien más retome nuestras tareas y trabaje en ellas, y posiblemente las complete de manera más efectiva que nosotros en nuestro estado actual. Necesitamos determinar si somos realmente insustituibles o si alguien más podría tomar nuestro lugar. La respuesta podría ser “soy insustituible, pero necesito salir de esta situación o me haré daño a mí mismo y a los demás si esto continúa”. Depende de nosotros revisar si nos estamos ayudando a nosotros mismos y a las organizaciones a las que servimos, o si los estamos perjudicando a ellos y a nosotros mismos al engañarnos pensando que esto está funcionando.

El tercer miedo tiene que ver con nuestro propio concepto personal de identidad y la memoria de nuestra comunidad. Si decidimos dejar de ser programadores, ¿borrará eso de alguna manera una parte de nuestra identidad? ¿Nuestra comunidad dejará de identificarnos como programadores y perderemos contacto con personas que se han convertido en amigos y colegas? Una vez más, este miedo es difícil de superar porque la programación puede ser una parte importante de nuestra identidad. Dejar de lado la programación nos puede llevar a sentir que nos estamos desprendiendo de una parte de nosotros mismos y de nuestra identidad. También existe el temor de que la gente deje de llamarnos para trabajos u otros proyectos de programación si decidimos tomarnos un descanso temporal. Si el descanso es temporal, ¿la gente recordará nuestras habilidades en programación cuando decidamos regresar?

Cada uno de estos miedos y preocupaciones son perfectamente válidos, pero pueden no ser ciertos. Podemos sentir miedo de que estamos malgastando nuestro tiempo como programador, pero la verdad es que cualquier proceso de aprendizaje no es en absoluto un esfuerzo malgastado. Podemos preocuparnos de cómo otras personas nos perciben o de cómo la empresa de la que formábamos parte continuará sin nosotros, pero la verdad es que no podemos controlar sus percepciones ni sus acciones. Lo que podemos controlar es nuestra participación en cada una de esas comunidades y nuestra propia percepción del tiempo y esfuerzo invertidos. Podemos determinar si una ruptura profunda de la programación sería

mejor que una salida gradual de nuestros compromisos. Podemos aclarar con las demás personas cual es nuestro estado actual como programador y determinar si este estado es temporal o permanente. La cuestión más importante es no permitir a los demás que nos convenzan de hacer algo que no queremos o que es perjudicial para nosotros. Si necesitamos parar nuestra actividad en programación porque estamos emocionalmente exhaustos y agotados, entonces debemos dejarlo claro a los demás que les estaremos perjudicando a ellos y a nosotros mismos si continuamos.

Las comunidades más maduras entenderán la necesidad de tomar un descanso y dejar de programar. Entenderán que tu bienestar mental y emocional es más importante que sus necesidad de que continúes y serán capaces de realizar lo que se necesite hacer y subsanar tu ausencia. Es normal y natural en las personas el cambiar de empresas y perseguir otras prioridades.

Lo más importante para recordar, es que es correcto el apagar esa parte de tu ser y dejar de ser programador. Si haces o no que un cambio permanente depende de ti y de tus deseos. Sentirse emocionalmente exhausto, sin inspiración y agotado es contraproducente para la práctica de programación: la programación es una tarea bastante difícil. Tomarse un descanso de la programación para explorar otros intereses es natural y no significa que seas menos programador por querer hacer algo diferente para recargarte de energía. Si descubres que eres más feliz cuando no estás programando, busca cualquier otra cosa que llame tu atención con todas tus ganas. Si decides volver a la programación después de estar fuera durante un tiempo, puedes regresar y retomar tu práctica de aprendizaje. Recuerda: nuestras vidas toman muchos giros y caminos diferentes. El mejor camino para ti es el que haces tú mismo, sin importar a dónde te lleve.

8 Epílogo

Es un estereotipo para un autor el decir que el libro que ha escrito es el que le hubiera gustado leer cuando se enfrentó a los temas presentados en el libro. Quizás es un estereotipo porque es cierto: este libro contiene los consejos que me habrían ayudado cuando comencé este viaje. Con demasiada frecuencia me he preguntado si he estado a la altura de los niveles que he creado para representar al programador ideal. Muchas veces he visto el éxito de mis compañeros y me pregunto si yo soy un programador ineficaz o deficiente en mi aprendizaje. ¿Qué estaba haciendo mal que otras personas hacen bien?

He llegado a la conclusión que el viaje de cada programador es único. Tu viajes te llevará por direcciones que serán diferentes a las que me llevó mi viaje. Tendrás experiencias que yo no puedo compartir y yo he tenido experiencias que difícilmente tu podrás replicarlas a menos que tengas una máquina del tiempo. Ninguna de nuestras experiencias es menos válida que las del otro, ni son más o menos válidas que las experiencias de otros programadores. Estas son nuestras experiencias. Las áreas de nuestro conocimiento son solo las áreas que hemos explorado hasta ahora. Siempre habrá brechas, pero podemos definir esas brechas como las áreas que aún no hemos explorado. Siempre hay más para explorar, y esa exploración es la parte divertida del viaje.

Ningún viajero puede estar en todos los lugares en todo momento. Deben viajar a cada destino tan rápido o tan despacio como sus medios de transporte les permitan y permanecer allí tanto como puedan antes de viajar a su siguiente destino. Viajan con los compañeros que encuentren en cada comunidad que puedan crear. Construyen relaciones y confían en ellos mismos y en los demás. Utilizan sus puntos fuertes para ayudar a otras personas y explorar y mejoran sus debilidades. Cada día continúan adelante. Como el viajero, a menudo debemos escoger nuestros destinos y compañeros. Podemos encontrar a aquellos, que como nosotros, están viajando por el mismo camino y que nos pueden ayudar en nuestro viaje.

Podemos intercambiar historias sobre nuestros éxitos y fracasos y experimentar cada día como otra etapa más en nuestro viaje.

Continuo mi viaje cada día y espero que como programador tu continúes tu viaje cada día mientras seas capaz. Puede que no estemos exactamente juntos en el mismo camino, pero tenemos la misma meta: hacer lo mejor que sabemos hacer en cada momento.

Te deseo lo mejor en todos tus viajes y espero volver a escuchar las historias de tus viajes cuando nos encontremos de nuevo.

9 Agradecimientos

Este libro no existiría sin las personas que me han acompañado en mi viaje, tanto docentes como colegas. Mi agradecimiento y aprecio a todos mis docentes en mis años de formación por brindarme sus mejores esfuerzos para enseñarme programación en sus diversas formas. Estoy en deuda con todos mis colegas y amigos programadores a lo largo de los años que compartieron sus conocimientos conmigo y confiaron en mí lo suficiente como para ayudarlos en el camino. También tengo la suerte de tener muchas comunidades que me ayudan a mantenerme, incluidas Michigan!/usr/group, PyOhio, Coffee House Coders y Ubuntu Michigan Loco. También a los que no encajan en estas categorías. Sepa que si hemos pasado algún tiempo discutiendo la programación u otros asuntos, nuestras discusiones son apreciadas profundamente.

También agradezco el trabajo de Leo Babauta de Zen Habits que me proporcionó las ideas de *mindfulness* y contenedores de atención focalizada. Han sido transformadoras en mi propio trabajo, como lo demuestra este libro. Me comprometí a pasar al menos 10 minutos cada mañana escribiendo cada sección, y los resultados son el trabajo que ves ante ti.

Gracias a los que me ayudaron directamente con este proyecto. Gracias a mi madre, Sharon Maloney, por ayudarme en la edición de este libro. Cualquier eerrorr que quede es responsabilidad del autor. Gracias a Beau Sheldon por revisar el capítulo sobre salud mental y por ayudarme a comprenderlo mejor y resaltar las áreas en las que lucha la gente. Gracias a mi amigo, David Revoy, por su increíble portada y por su inspiración durante todo el proyecto. Gracias a Esteban Manchado Velázquez por agregar CSS y limpiar la versión HTML del texto. Gracias a los lectores beta por sus valiosos comentarios y opiniones en repositorio git en Framagit, incluidos (en orden alfabético por identificador o nombre): Brendan Kidwell, D. Joe Anderson, David Revoy, Eric Hallam, Jer Lance, Matthew Piccinato, Matthew Balch, Midgard, Nicholas Guarracino, RJ Quiralta, Valvin y Wilhelm Fitzpatrick. Gracias a Paco Esteban por los arreglos de edición.

Mi más profunda gratitud es para mi mujer JoDee y mis padres por su apoyo y por creer en mí. Las palabras no pueden expresar el amor y el agradecimiento que tengo por ellos.

10 Mi viaje

Mi viaje como programador comenzó cuando estaba en la escuela primaria. Me interesé en las computadoras después de leer sobre ellas en la *World Book Encyclopedia* y esperaba trabajar con ellas algún día. Lo que no me di cuenta fue que esas enciclopedias estaban desactualizadas y solo mostraban las computadoras centrales y las minicomputadoras más grandes y costosas de la década de 1960 y no las microcomputadoras más modernas que se introdujeron a fines de la década de 1970. Cuando me di cuenta de que una Apple era una microcomputadora y que estaba diseñada para el mercado doméstico, comencé mi búsqueda para obtener una computadora propia (es decir: comencé a dar pistas no muy sutiles a mis padres de que quería una computadora). Revisé revistas como *Popular Computing* y *Byte Magazine* en busca de la computadora adecuada, desde el Commodore VIC-20 y el Sinclair ZX-80 hasta el Radio Shack TRS-80 Model III (incluso el Rockwell AIM-65 o el Heathkit H89 habrían servido. Por aquel entonces no era quisquilloso). Mi padre me llevó a tiendas donde las vendían y me maravillé de la variedad de máquinas que había allí (y probablemente puse nerviosos a algunos vendedores mientras tecleaba y tocaba las máquinas nuevas y bastante caras). Finalmente, mi padre compró una computadora Atari 400 con unidad de cinta y comencé a aprender programación BASIC en serio. Casi al mismo tiempo, mi escuela abrió un “laboratorio de computación” con tres máquinas Commodore PET 4032 (completas con unidades de disquete), y me encontré pasando cada momento que podía con esas máquinas. En la escuela secundaria tomé dos cursos de programación, uno en BASIC y el otro en Pascal (que fue mi primera exposición a los lenguajes de procedimiento y los conceptos básicos de la informática). En la universidad me especialicé en Ciencias de la Computación con una Licenciatura en Ciencias e hice todo lo posible para mantenerme al día con todas las cosas que intentaron enseñarme. Desafortunadamente, no era un gran estudiante (especialmente en matemáticas). Luché y luego abandoné mi clase de compiladores, y sentí que me estaba quedando atrás en comparación con otros estudiantes que tenían éxito. La mayoría de nuestras clases usaban Pascal, con el que me estaba familiarizando, pero había

algunas clases que usaban COBOL, Ada, SNOBOL, C y lenguaje ensamblador. Me gradué con notas modestas y regresé a casa.

A lo largo de mi carrera, he cruzado la brecha entre la administración de sistemas y la programación. Linux era similar a las máquinas SunOS que admiraba en la universidad, así que pasé a usar eso como mi sistema operativo principal alrededor de 1995. En mis primeros trabajos me encargaron el mantenimiento de varios tipos de computadoras: PC de escritorio, máquinas basadas en UNIX y de manera ocasional de las copias de seguridad de las máquinas VAX. No fue hasta que uno de mis puestos necesitaba un sitio web que agregué más programación a mi currículum. La programación de sitios web en la década de 1990 fue donde realmente comencé a aprender y comprender Perl, SQL, bases de datos relacionales y HTML. La web era tan nueva en la década de 1990 que todas las personas en nuestros proyectos estaban aprendiendo al mismo tiempo. Aproveché mi conocimiento de Perl en algunos otros trabajos y proyectos haciendo programación basada en web. Perl en la década de 1990 era un lenguaje donde los conceptos básicos eran simples de aprender pero el lenguaje podía manejar ideas y estructuras de datos realmente complejas. Perl y CGI facilitaron la incorporación en una página web de algo que tuviera un poco de interactividad. Donde Perl se vuelve complejo es la sintaxis de cosas como las expresiones regulares y la tendencia de los programadores de Perl a valorar el código que realiza múltiples acciones en la misma línea. La comunidad de Perl también valora el código inteligente, lo que me llevó a preguntarme en varias ocasiones si era lo suficientemente inteligente como para ser un programador de Perl.

Una de las empresas en las que trabajé decidió migrar un sistema Perl a un entorno basado en Java. Analizaron las habilidades del equipo de desarrollo existente y decidieron que necesitaban externalizar el proyecto a otra empresa. Esta era una tendencia común a principios de la década de 2000 por razones que están fuera del alcance de este libro. Esto me dio un primer contacto como líder de un equipo. Sé que muchos programadores migran a roles administrativos, pero en ese momento sentí que no había explorado completamente mi potencial de programación. Me senté en varias ocasiones e intenté aprender Java, pero nunca me llamó la atención. El desarrollo web Java siempre lo sentí más engorroso que los scripts CGI de Perl que creé.

Tampoco ayudó que enviáramos archivos `.war` a un sistema Tomcat, que parecía estar compuesto por una gran cantidad de metadatos de configuración y muy poco código. Esto es a lo que me refería cuando hablé de estar bien con renunciar a aprender algo, a veces lo que tratamos de aprender es más una tarea rutinaria. Tener algo que es una tarea rutinaria no va a proporcionar una buena experiencia de aprendizaje.

Fue en ese momento cuando comencé a aprender Python por mi cuenta usando Pygame. Me lancé al participar en mi primera competición Pyweek. Pyweek es una competición de una semana en la que la gente crea un juego completo desde cero. Fue un desafío, pero también fue una de las experiencias más gratificantes que he tenido en la programación. Creé un juego llamado “Busy Busy Bugs” que se podía jugar en un momento en que realmente no sabía lo que estaba haciendo. En muchos sentidos, estaba aprendiendo a nadar arrojándome al proverbial fondo profundo. No estaba en peligro, pero el deseo de hacer algo al final de la semana me guió de maneras que no creía que fueran posibles.

A medida que la posición de líder técnico continuaba, me encontré con ganas de hacer otra cosa. Me entrevisté en varios lugares y fui contratado en Sourceforge como miembro del Grupo de Operaciones de Sistemas. Sourceforge fue una experiencia increíble para mí. Soñaba con trabajar para una empresa de código abierto, y pocas empresas de código abierto eran tan bien consideradas como Sourceforge y Slashdot en la comunidad de código abierto, pero mis inseguridades y el “síndrome del impostor” entraron en acción. ¿Sería capaz de cortarlo? ¿Me contratarían solo para darse cuenta de que habían cometido un error y que no era tan bueno como decía ser? Yo era amigo y había ido a la escuela con varias de las personas que trabajaban en Sourceforge/Slashdot, así que me preguntaba cuales eran sus motivaciones para contratarme. ¿Me contrataron como una broma elaborada o me contrataron porque varias personas en la empresa me conocían? Todos estos pensamientos pasaron por mi cabeza mientras trabajaba allí. No ayudó que mi posición fuera principalmente administración de sistemas a un nivel en el que no tenía experiencia. También había un componente de programación en el puesto, pero constantemente sentía que estaba muy por encima de mi cabeza. Vivía con el temor constante de que me descubrieran y de que el trabajo que quería ya no estuviera disponible para mí. De

acuerdo, aprendí mucho y tuve unos jefes muy amables en Sourceforge, pero vivía con pavor cada vez que mi gerente me consultaba, porque temía que la conversación solo resaltara que se suponía que yo no debía estar allí.

Me encantaría decir que tuvo un final feliz y que mis temores eran infundados, pero lamentablemente me despidieron de ese puesto debido a restricciones presupuestarias. Estoy agradecido por las oportunidades que tuve allí, los amigos que hice y las experiencias que tuve, pero estaría mintiendo si no dijera que el despido vino con una mezcla de tristeza y alivio. Tristeza por no poder volver a tener un trabajo genial como ese y alivio por poder dejar de lado por el momento esos sentimientos del síndrome del impostor. En muchos sentidos, crecí a partir de la experiencia de trabajar en Sourceforge y aprendí mucho sobre mí y mis capacidades mientras trabajé allí, pero también fue el puesto en el que sentí mi síndrome del impostor en su máxima expresión.

Más tarde me contrataron para un puesto a tiempo completo haciendo programación Python. Este empleo me permitió fortalecer mi oficio como desarrollador de Python. Allí creé muchos proyectos interesantes y seguí aprendiendo más sobre Python. Me ayudó a recuperarme y ampliar mis habilidades. La gente en este trabajo fue muy comprensiva y amable. Hubo momentos en los que se volvió estresante (todos los trabajos parecen tener estrés), pero en general fue una experiencia positiva.

Lamentablemente, también me despidieron de ese puesto (el dinero apesta, ¿sabes?).

Empecé mi búsqueda de trabajo en serio y fui a un montón de entrevistas. Si bien todas las personas en las entrevistas parecían impresionadas con mis habilidades y mi carrera, caí en una de dos categorías: no encajaba bien en el puesto o no tenía suficientes conocimientos en las áreas que buscaban. Me encontré haciendo pruebas cronometradas de escribir código que parecían estar probando si me estresaba fácilmente en lugar de las habilidades de de escribir código que pudiera tener. Me senté en sesiones en las que escribía código con figuras sombrías que gritaban comandos y requisitos mientras hacía todo lo posible por seguirlos. Hice acertijos de matemáticas y problemas de lógica (lo cual es horrible si no eres bueno ni en matemáticas ni en problemas de lógica). Los primeros indicios

prometedores se convirtieron más tarde en cartas de rechazo (suponiendo que me respondieran), y la desesperación se apoderó de mí cuando me enfrenté a la perspectiva real de que tendría que dejar la programación si quería ganarme la vida. Las visiones de regresar a los comienzos de mi carrera me llenaron de pavor. Parecía que ya nadie quería arriesgarse conmigo y no podía competir con tantos programadores nuevos que no habían cometido los errores que yo había cometido en mi carrera.

Durante ese periodo registré el dominio web themediocreprogrammer.com. Si yo iba a ser un desastre entonces podría aceptarlo.

Afortunadamente, he tenido una comunidad de amigos y compañeros programadores que me han ayudado. En mi trabajo actual estoy contratado por uno de estos amigos al que ayudo con las tareas de programación. Ese puesto surgió al asistir a PyOhio (una conferencia de programación) que se celebra todos los años. En todas mis luchas, he tenido la suerte de tener allí a una comunidad que me ha ayudado. Es por eso que creo que las comunidades son tan geniales: nos brindan una red de personas que de otra manera no tendríamos, y nos ayudan a superar nuestros triunfos y nuestras luchas.

Soy una colección de todas estas experiencias. Todas ellas me hacen quien soy. A veces me pregunto si debería haber tomado un camino diferente o haber hecho algo diferente, pero ese es un ejercicio inútil. Lo mejor que puedo hacer con estas experiencias es aprender de ellas y seguir adelante. Cada día trabajo para mejorar y superarme. Cada día cometo nuevos errores, pero todo eso es parte del proceso de aprendizaje.

Mi viaje continua.