No lo desarrolles.

UNA GUÍA PARA PROFESIONALES DE LA TECNOLOGÍA CÍVICA / TECNOLOGÍA PARA EL DESARROLLO

Esta guía busca ayudarte a evitar malos proyectos, estructurar bien a tu equipo y a aprender, crecer y lanzar productos más rápido.

Luke Jordan / Grassroot & MIT Governance Lab / **2021**

GRASSROOT

MITGOV/LAB

Cita recomendada:

Jordan, Luke. 2021. “No lo construyas: Una Guía Para Profesionales De La Tecnología Cívica / Tecnología Para El Desarrollo”, Grassroot (ZAF) & MIT Governance Lab (EEUU).

Introducción

Luke Jordan es el fundador y director general/técnico de Grassroot, una organización de tecnología cívica con sede en Sudáfrica, y el practicante profesional de 2021 en el MIT Governance Lab. *Contacto: luke.jordan@grassroot.org.za.*

Grassroot

Grassroot es una plataforma tecnológica creada para entornos con bajo ancho de banda y datos escasos que permite la mensajería inteligente a través de mensajes de texto (www.grassroot.org.za).

MIT Gov/Lab

MIT Governance Lab es un grupo de politólogos que se centra en la innovación de la participación ciudadana y la capacidad de respuesta del gobierno (www.mitgovlab.org).

Gracias

Gracias a todo el equipo de Grassroot por sufrir las lecciones aprendidas en la guía, incluyendo a Katleo Mohlabane, Busani Ndlovu, Mbalenhle Nkosi, March Ratsela, Zinhle Miya, y Paballo Ditshego. Y, gracias a Tiago Peixoto, Arjun Khoosal, Lily L. Tsai, Alisa Zomer, y Maggie Biroscak por sus contribuciones a varias versiones de esta guía. Ilustraciones y diseño gráfico por Gabriela Reygadas y Susy Tort.

Gracias a ILDA por la traducción al español.

ILDA

PREFACIO

Esta guía es para los equipos o líderes que están pensando en construir o están construyendo un proyecto de “tecnología cívica”, es decir, la tecnología que ayuda a la ciudadanía a participar en el gobierno de manera más efectiva. Es el resultado de los cuatro años que pasé construyendo Grassroot, una plataforma de tecnología cívica en Sudáfrica.

En esta guía me concentré en lo práctico. Elegí los temas reflexionando sobre las cosas que la gente me ha pedido consejo a lo largo de los años; sobre lo que me hubiera gustado saber cuando empecé, sobre los consejos que me resultaron más valiosos en un principio, y sobre algunas de las cosas que salieron mal en el camino.

Tener un software o una idea de software no garantiza que ese producto debe ser construido, las organizaciones o equipos tienen que tener precauciones para identificar cuando su idea o producto no debe desarrollarse. Esto es difícil de identificar cuando a tu alrededor las personas siempre están intentando desarrollar nuevas aplicaciones, además de la presión generada por los medios que insisten en la “tecnología para el bien común”. Como prueba de estos impulsos, podemos observar que durante los últimos años, varias investigaciones han señalado el efecto limitado (si es que hay alguno) de proyectos tecnológicos con buenas intenciones aunque sin mucho rigor (la famosa frase “¡hay que hacer una app para eso!”). Y a pesar de los resultados de esas investigaciones, las aplicaciones se siguen desarrollando sin mucha reflexión.

Observar estos procesos ha sido mi motivación al escribir esta guía. Aunque creo que la tecnología puede ayudar a los ciudadanos a ejercer su poder y hacer que el Estado sea más responsable y receptivo, es mucho más importante utilizarla en el momento correcto. También he visto de cerca cómo la presión del pensamiento contemporáneo, las dinámicas de financiamiento y la búsqueda de prestigio nos empujan a desarrollar lo que no se debe desarrollar, con equipos que no saben cómo hacerlo. El tono de este texto no es el típico de las guías académicas: mi enfoque es describir el proceso de forma honesta y auténtica, con la esperanza de aportar una idea más clara de lo que involucra “desarrollar una aplicación” y de cómo hacerlo bien, o (mejor aún) no hacerlo si no es necesario.

No lo desarrolles

La guía comienza con la selección de proyectos, incluyendo una explicación de por qué creo que es mejor no hacer ningún proyecto si no es necesario. Después revisamos la estructura del equipo y la importancia de contar con un director técnico senior de tiempo completo [*o chief technology officer (CTO)*]. Posteriormente, nos enfocamos en los plazos del proyecto, haciendo énfasis en el lanzamiento acelerado del producto y la paciencia que se requiere para llegar a la madurez, sobre todo en la búsqueda de la funcionalidad del producto. Otro tema importante relacionado con la paciencia es evitar las métricas de vanidad[1] , lo cual también exploramos en esta sección. Después, hablamos sobre los procesos de contratación, el rol del director de tecnología, los contratistas senior, los diseñadores y los ingenieros.

La sección más larga (por mucho) es la referente a la contratación del personal. Esta es la única cosa que se considera crítica en todas las secciones, tanto por los fundadores como por los inversionistas y los directivos, en todos los sectores. También es el campo en el que creo que acerté la mayor cantidad de veces, por razones que puedo explicar de una manera que creo que será útil.

1 Las vanity metrics (métricas de vanidad) son métricas que presentan a tu tecnología de forma positiva, pero que no reflejan adecuadamente la realidad del crecimiento. Más información sobre estas métricas en la sección Usuarios vs. Métricas de Vanidad.

SÍNTESIS

Solo tienes que recordar esto...

Si puedes evitar desarrollarlo, no lo desarrolles; si tienes que desarrollarlo, contrata a un director de tecnología, lanza rápido el producto aunque no sea perfecto y con el tiempo aprenderás nuevas lecciones que van a ayudar a mejorarlo. En cualquier caso, recurre a un equipo de confianza, construye de forma rápida y sencilla, y establece vínculos con tu audiencia lo más rápido posible.

No lo desarrolles

Si tienes que desarrollarlo:

- *La subcontratación (o outsourcing) es una buena idea para hacer algo rápido, pero es una pésima estrategia.*
- *Contrata a tu personal de tiempo completo con prudencia, garantizando salarios con los que puedas mantenerlos en el equipo e invertir en sus habilidades a través del tiempo.*
- *Acércate a las personas que usarán la tecnología desde el inicio y busca un equipo al que le importe la participación comunitaria.*
- *La adaptabilidad y el aprendizaje rápido son criterios fundamentales en todos los puestos.*
- *Fija un presupuesto que te permita empezar rápidamente, pero con el que puedas seguir iterando a lo largo del tiempo.*

Para este tipo de guías, una regla común es nunca dar detalles específicos para evitar crear expectativas falsas. Otra regla común en el desarrollo de software es decir “se va a tardar lo que se tenga que tardar”. Voy a romper ambas reglas, y decir que con un buen equipo de 3-4 personas, que siga las buenas prácticas, deben ser capaces de lanzar un producto moderadamente complejo en unos 3 meses, y obtener algún tipo de estabilidad y madurez en 12 meses. Después de esto, por supuesto, se vuelve a empezar — si tienes una audiencia. Si no la tienes, detente a los 6 meses, o mejor aún, a los 4 meses, o mejor aún—**no lo desarrolles.**

Fundamentos técnicos: opta por el código abierto y elige un lenguaje de desarrollo que sea popular.

No hay nada más importante que un aprendizaje temprano rápido, y la dependencia en métricas de vanidad interrumpe ese aprendizaje.

¿ES NECESARIO DESARROLLAR ESTE PROYECTO?

↳ *PROBABLEMENTE NO*

El problema central del software es que se puede desarrollar cualquier idea

El problema central del software es que se puede desarrollar cualquier idea. La naturaleza y la física imponen limitaciones sobre las ideas materiales. En cambio, puedes terminar un proyecto de software y hasta el momento de implementarlo darte cuenta de que es fundamentalmente defectuoso. Peor aún, estamos tan acostumbrados a la mala tecnología que nadie se sorprende. Si construyes un edificio espantoso en medio de la nada, puede que se hagan películas sobre lo que hiciste; si desarrollas una aplicación inútil que nadie usa, sólo tendrás que citar una métrica engañosa en un informe para inversionistas y no le importará a nadie. De la misma manera, **si construyes un gran edificio** en un lugar sensato tal vez no habrá mucho interés; aunque si desarrollas una aplicación que no es mala y que las personas usen más allá de la campaña publicitaria de lanzamiento, es probable que te llamen para aparecer en media docena de listas de líderes emprendedores.

El mejor método es adoptar un principio simple:

¡No la desarrolles!

Cuando alguien diga,

Solo di que no.

Cuando un inversionista te diga,

Solo di que no.

Cuando leas otro artículo o veas otra TEDx sobre alguien que cree que su aplicación ha logrado algo, citando cifras no verificadas y sin sentido, y una voz interior te diga:

Solo di que no.

¿Esto quiere decir que el resto de esta guía es inútil?

Esperemos que sí. Pero en realidad, en algún momento alguna idea puede parecer tan buena que la idea de **"no desarrollarlo"** puede parecer ilógica. En ese momento, pregúntate esto:

¿Ya hay personas que intentan hacer lo que se supone que la tecnología les ayudaría a hacer?

En caso de ser así, ¿qué es lo que hacen ahora, y estás seguro de saber por qué no funciona? ¿Por qué crees que la tecnología va a hacer alguna diferencia en algo que no está funcionando aún?

Si no es así, ¿qué diferencia habría en contar con esta tecnología? ¿Por qué alguien que no quería hacer algo va a cambiar de opinión sólo porque existe una nueva manera para hacerlo?

Por supuesto, es fácil dar respuestas engañosas a estas preguntas para justificar las ganas de desarrollar algo, y eso es lo que pasa la mayoría de las veces. Pero supongamos que has respondido con honestidad y resulta que, por ejemplo, la gente intenta hacer lo que se supone que la tecnología les permitiría hacer y no funciona por algún problema fundamental.

Puede ser una gran tentación utilizar la tecnología para intentar resolver ese problema. Pero, una vez más, **¡no lo desarrolles!** Primero hazte las preguntas anteriores sobre ese problema fundamental que has descubierto. Por ejemplo, antes de construir un programa que ayude a un gobierno a percibir un fenómeno (la violencia o los apagones), pregúntate: ¿No hay nadie intentando informarles sobre esto? Si se les ignora, ¿por qué se les ignora? Si se debe a los desequilibrios de poder, tu tecnología bien intencionada ¿*realmente* aportará algo? ¿O podría ser irrelevante, o incluso empeorar el problema al permitir a las personas con poder fingir

No lo desarrolles

(ante sí mismas y ante otros actores) que están haciendo algo? Si nadie asiste a las reuniones del gobierno, ¿es porque no saben de las reuniones, o porque cuando asisten nadie les hace caso? Estos ejemplos podrían multiplicarse de forma casi interminable.

Si después de preguntarte todo esto, encuentras un problema en el que está claro que a) hay personas que intentan hacer lo que se supone que la tecnología les permitiría hacer, pero no funciona; b) no funciona principalmente *debido a algún problema que la tecnología puede resolver*, y c) los motivos para esto son *evidentes* y *seguros*... entonces podría ser el momento de empezar a considerar la posibilidad de desarrollar algo.

OKAY, HAY QUE DESARROLLARLO. PERO, ¿QUIÉN LO VA A DESARROLLAR?

PROYECTO Y ESTRUCTURA DEL EQUIPO

Supongamos que has seguido los pasos anteriores y que has descubierto un proyecto tecnológico que realmente vale la pena. La siguiente pregunta es: ¿dónde encontrarás y cómo vas a organizar a las personas que van a diseñar, codear, lanzar, monitorear e iterar el proyecto?

La mayoría de las organizaciones de sectores no técnicos tendrán el impulso de subcontratar (outsource) la mayor parte del trabajo. La subcontratación (o outsourcing) es una buena idea para hacer algo rápido, pero una pésima estrategia. Subcontratar puede ser muy rentable cuando se trata del trabajo de desarrolladores y diseñadores con mucha experiencia, pues proporciona una gran flexibilidad. De esta manera pueden participar en los proyectos sin consumir el presupuesto en tiempo muerto y las nuevas plataformas de freelance permiten encontrarlos para necesidades muy específicas. Sin embargo, la subcontratación total de un equipo hace que el éxito de todo el proyecto dependa en gran medida del contratista que encuentres, y arriesga el proyecto. Si uno de los *freelancers* no encaja bien, puedes sustituirlo y encontrar a alguien más, a menudo sin interrumpir al resto del equipo. Si todo el equipo está subcontratado y no funciona, tendrías que reemplazar al equipo, detener la mayor parte del trabajo e incluso podrías enfrentarte a una disputa legal por romper el contrato.

La sub-contratación es una buena idea para hacer algo rápido, pero una pésima estrategia.

A menos que una organización cuente con un **Director de Tecnología (CTO)** competente (más sobre esto a continuación) lo más probable es que no se elija a un contratista adecuado. Al

No lo desarrolles

no tener un experto en el equipo, se pierde la capacidad de juzgar los méritos técnicos de las propuestas o entender los componentes técnicos, especificaciones y requisitos de un proyecto. Una contratación sin especificaciones técnicas claras puede llevar a una entrega deficiente de la cual ya hizo un acuerdo contractual. Por otro lado, demasiadas especificaciones pueden llevar a una peor entrega, porque se pierde la flexibilidad de adaptarse mientras se construye el producto, al darle prioridad a las especificaciones marcadas en el contrato.

Puede que un programador de confianza contribuya a las especificaciones, pero suele ser muy obvio cuando el redactor principal de una solicitud de propuesta (*request for proposal*, o RFP por sus siglas en inglés) u otro documento de licitación no tiene conocimientos técnicos. Existe toda una industria de consultores que realizan estas solicitudes de propuestas sin conocimientos técnicos, y no son necesariamente quienes quieres que se encarguen de un proyecto que decidiste que debes desarrollar (debido a que, si el proyecto es opcional, no debes desarrollarlo.)

Una vez que la solicitud de propuestas se haya publicado, sin una red de contactos técnicos, ¿cómo vas a encontrar a contratistas a quienes invitar? Y una vez que los contratistas oferten, ¿cómo los vas a evaluar, tanto en la selección como en los objetivos iniciales? ¿Realmente va a hacer todo eso tu programador de confianza que hizo algunas recomendaciones en la solicitud? Si no es así, tendrás que limitarte a seleccionar basándote solo en un portafolio y a la experiencia que la persona dice que tiene. Debido a la frecuencia con la que las organizaciones sin conocimientos técnicos

seleccionan a malos contratistas técnicos, contratar sobre la base de un portafolio sin tener la capacidad de hacer preguntas técnicas es peor que una selección al azar, porque estarás seleccionando sobre la habilidad que el contratista tiene para venderse y eso está, en todo caso, inversamente correlacionado con la capacidad técnica.

De vez en cuando, alguna organización tiene suerte y un proyecto limitado encuentra al contratista perfecto y funciona. Pero tener una estrategia con tantas probabilidades de fallar, a menos de que tengas mucha suerte, es una estrategia terrible.

Director de Tecnología (CTO)

Algunas organizaciones pueden preocuparse por cómo van a cubrir el costo de un director de tecnología entre tantos proyectos. Si esto te preocupa, **no desarrolles tu proyecto**: tu organización no está preparada para mantener un buen producto tecnológico. Esa preocupación significa que asumes que la tecnología se desarrolla como un proyecto aislado, que se completa cuando se lanza, y que no necesitará un trabajo continuo para iterar y hacer mejoras. Y si ese es el caso, entonces no entiendes bien el problema (vuelve al paso 1), o no entiendes el uso adecuado de la tecnología, y el proyecto está destinado a fallar desde un principio.

Evidentemente, los recursos pueden cambiar con el tiempo. Pero al menos durante varios años después de emprender un proyecto tecnológico útil, éste requerirá al menos la mitad o más (casi siempre más) del tiempo de un buen director de tecnología. Al igual que no deberías adoptar un perrito si nadie en tu casa quiere pasearlo o darle de comer cuando ya no sea tan tierno, no deberías embarcarte en un producto tecnológico a menos que tengas un CTO capacitado que pueda hacerse cargo de él cuando crezca.
En el caso excepcional de que un proyecto esté tan avanzado que un director de tecnología tenga tiempo de sobra, puede trabajar en actualizaciones tecnológicas internas (por ejemplo, al iniciar la pandemia, muchas organizaciones se dieron cuenta de problemas internos en los que debieron haber trabajado desde hace años), o colaborar en tareas en las que se comparten costos con otros socios. En la mayoría de las organizaciones, un buen director de tecnología siempre encontrará la manera de ser útil.

Personal técnico de tiempo completo

Para el resto del equipo, el tipo de empleado (y sus habilidades) deben de escogerse basado en el producto. Contrata a tu personal de tiempo completo con prudencia, garantizando salarios con los que puedas mantenerlos en el equipo e invertir en sus habilidades a través del tiempo. Es importante dejar suficiente presupuesto para contratar a personas excepcionales, en un corto plazo, para que realicen partes valiosas del proyecto o que utilicen tecnología especializada. Por ejemplo, puedes hacer un equipo con un programador junior de front-end con un contratista *senior* de *front-end*, por un mes, para que juntos establezcan la estructura del código y los componentes básicos iniciales, y que esta experiencia sirva como asesoría al programador junior, para que pueda desarrollar el resto.

Contar con personal de tiempo completo durante un periodo largo de tiempo es una inversión que vale la pena, sin importar el nivel de experiencia que tengan. Los miembros de la organización van asimilando los conocimientos, y los retienen a lo largo del tiempo. Los contratistas *senior* aportan un conocimiento profundo sobre su especialidad; los programadores internos aportan un profundo conocimiento del sistema; tener uno sin el otro crea un riesgo de fracaso, como un hospital que intenta depender exclusivamente de doctores en su personal, o carece de ellos por completo.

Hay una gran variedad de combinaciones, pero la idea es optimizar la continuidad en la base, aplicar habilidades extremadamente elevadas donde importa (aunque sea en plazos cortos) y mantener un buen margen de error.

Contrata a tu personal de tiempo completo con prudencia, garantizando salarios con los que puedas mantenerlos en el equipo e invertir en sus habilidades a través del tiempo

Definiciones:

Stack tecnológico: La combinación de tecnologías que componen un producto o proyecto (que, metafóricamente, se apilan unas sobre otras para crear algo útil o divertido).

Un ejemplo: una buena estructura de equipo puede consistir en un programador junior sin mucha experiencia, un experto técnico en las partes más importantes **del *stack***, y un grupo de freelancers *senior* especializados en una parte específica del *stack*, que se incorporan durante algunos días o semanas, dejando un presupuesto de reserva para ayuda de emergencia, si se considera necesario. Del mismo modo, sólo para ilustrar, una estructura de equipo mala podría consistir en tres programadores generales de nivel medio, con experiencia en aspectos no adecuados del *stack*, y no tener un fondo de emergencias (Figura 1). La única característica en común es que, independientemente de cómo se forme el equipo, es probable que el proyecto fracase si no se tiene un buen director de tecnología.

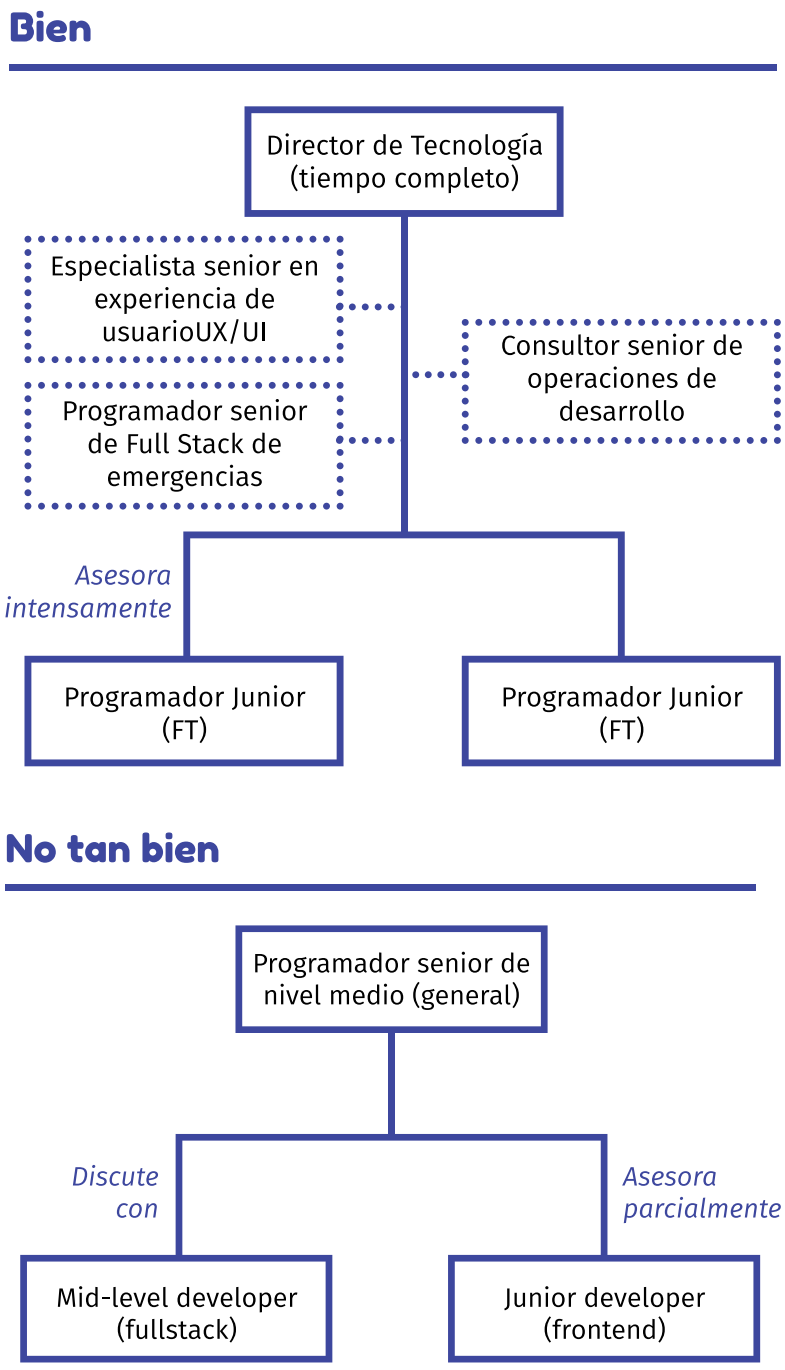


FIGURE 1 →

Conectando a los programadores, la experiencia del usuario/a y el personal de campo

Una última nota sobre la composición del equipo: se necesitan personas que puedan desarrollar la tecnología, pero también se necesitan personas que entiendan el contexto en el que se utilizará la tecnología, es decir, los países, las comunidades, las organizaciones y la demografía de la audiencia.

Parte de esa comprensión debería construirse con técnicas de "experiencia de usuario/a" (UX), a cargo de desarrolladores de UX calificados (véase en la sección Contratación, más abajo). Con el tiempo, se podrá justificar la contratación de un especialista en UX de tiempo completo. Pero, sobre todo al principio, es poco probable que cuenten con una demanda que justifique la contratación de un especialista en UX de tiempo completo, y probablemente no tengan el presupuesto necesario para tener uno a su disposición. Cuanto más entienda el equipo base sobre cómo se construyen las funciones (e, idealmente, hayan recibido formación básica en técnicas de UX), más podrán adaptar sus observaciones a lo que sea más útil para el equipo de desarrollo. De forma similar, mientras más entienda el equipo de desarrollo sobre lo que ocurre en el campo en el que sucede el problema, es más probable que propongan nuevas ideas o direcciones alternativas para el producto que se construye (o que reflexionen sobre lo que se está haciendo).

Dicho esto, entender bien el contexto no debería ser una regla rígida, especialmente en las contrataciones en puestos *senior*. Por lo general, la adaptabilidad y la velocidad de aprendizaje son criterios fundamentales en todos los puestos. Un empleado flexible y que aprende rápido puede diseñar o construir para un contexto nuevo, incluso si no tiene mucha experiencia en el campo. Por el contrario, un empleado que no es flexible ni aprende rápidamente, puede tener un conocimiento anticuado del contexto, ser lento cuando se necesite actualizar su conocimiento y puede tener resistencia a los puntos de vista que difieren de los suyos.

La adaptabilidad y el aprendizaje rápido son criterios fundamentales en todos los puestos

En el caso de los trabajos para principiantes, se puede y se debe dar mucha importancia al contexto, dando prioridad a los jóvenes programadores de las comunidades para las que se está desarrollando la tecnología. Según un antiguo dicho de Confucio, "las calles están llenas de sabios" — las calles también están llenas de programadores, siempre y cuando te interese encontrarlos. Vale la pena examinar los CVs para encontrar a

alguien con las capacidades y la mentalidad adecuadas, pero también con una experiencia de vida que pueda enriquecer las discusiones del equipo.

Sin embargo, nadie podrá adaptarse o aprender si no está en contacto regular con el contexto del proyecto, o con los comentarios directos de la audiencia potencial o real. Los programadores con una experiencia más tradicional, típicamente de equipos grandes o corporativos, pueden (al menos al principio) sentirse incómodos pensando en el contexto, ya que en las empresas tradicionales escribir código suele ser algo muy distinto a pensar en su uso. En los equipos pequeños (y en algunos casos, en los más grandes), todos deberían estar pensando en las necesidades de su audiencia.

En la tecnología cívica, eso significa establecer un vínculo firme entre el equipo de relaciones con la comunidad y el equipo de desarrollo. Pero eso también requiere contar con un equipo de campo o comunitario que esté profundamente inmerso y que realmente comprenda a la audiencia a la que se dirige el producto.

Invierte en la creación de vínculos entre los equipos de campo y los programadores hasta que la información fluya de forma natural entre ellos. Haz que los programadores acompañen al equipo de campo en algunas ocasiones; invita al equipo de campo a contribuir a un debate sobre características técnicas entre programadores. Cuando un programador venga y pregunte “no estoy seguro de esto”, *si es apropiado*, envíalo directamente a un miembro del equipo de campo; y viceversa, si un miembro del equipo de campo regresa preguntando “¿podemos hacer que la aplicación haga esto?”, y *tiene sentido*, dirigelo con el programador que puede implementarlo. Hay un balance frágil en esta relación — el CTO, o los directivos en general, necesitan mantenerse informados, y este tipo de actividades pueden ser ser poco productivas si no existe una buena relación entre los equipos. Sin embargo, sí es posible encontrar el equilibrio adecuado (Figura 2).

En Grassroot, empezamos con una estructura relativamente cargada a nivel *senior*, con un exceso de contratistas generales. A lo largo del tiempo, adoptamos una estructura de 1 ó 2 programadores junior a la vez, que en ocasiones permanecían en el equipo durante varios años, complementados por especialistas *senior* contratados durante periodos cortos de tiempo para realizar tareas específicas. Por mi parte (actuando como director de tecnología) me ocupaba personalmente de revisar el código de los especialistas y de asesorar a los programadores junior. Nuestro equipo de campo tenía instrucciones de pasar dos tercios de su tiempo con la comunidad y el resto de su tiempo reunirse con los programadores y comunicarles lo que habían observado. Complementamos esta estructura con especialistas en experiencia de usuario/a (UX), quienes se integraron al equipo de campo para observar el comportamiento de nuestra audiencia. Esta estructura también nos permitió incorporar con flexibilidad a otras personas talentosas, por ejemplo, a través de los programas de “informática para el bien común” que han desarrollado algunas universidades (en nuestro caso, trabajando con Stanford).

No lo desarrolles

Artificial

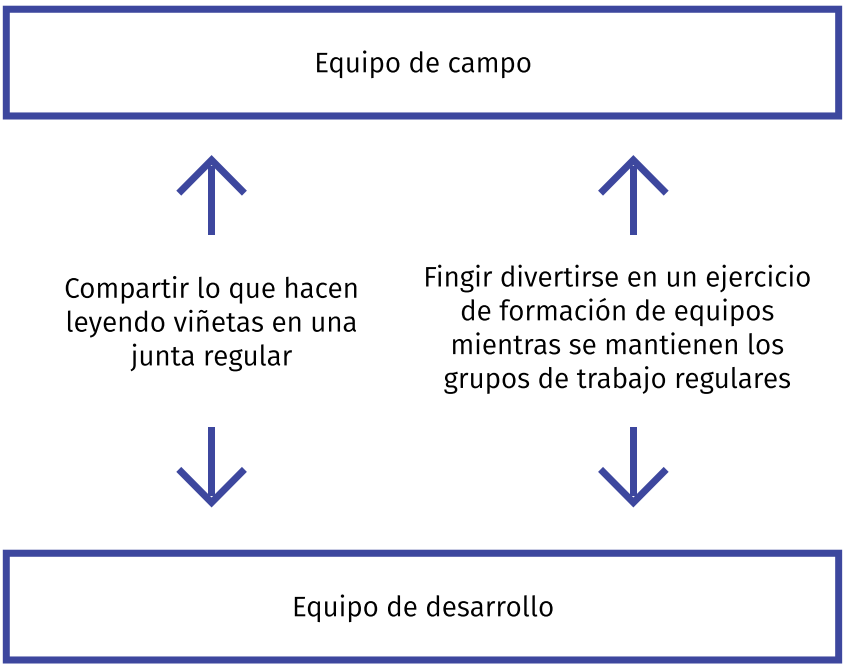
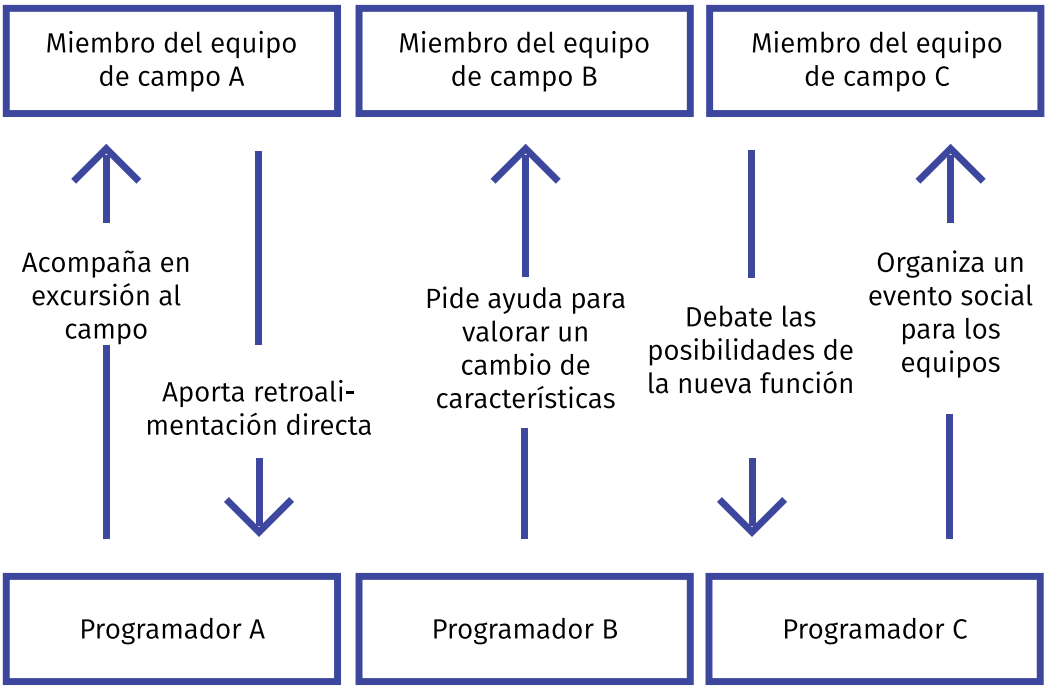


FIGURA 2

Orgánico



CONTRATACIÓN DE PERSONAL

Director de Tecnología (CTO)

La falta de claridad puede ser un obstáculo al contratar un Director de Tecnología (CTO). En las empresas grandes, la función del CTO suele implicar la supervisión de procesos, del personal directivo intermedio y de las contrataciones. Los buenos directores de tecnología se involucran en el desarrollo de vez en cuando, aunque principalmente supervisan una serie de procesos para el desarrollo. Este puesto es diferente en una startup, una organización sin fines de lucro o incluso una unidad gubernamental de servicios digitales. En estos casos, el director de tecnología desarrolla, y debe tener el gusto por desarrollar cosas.

Un ejemplo del perfil de la persona adecuada podría ser el de un desarrollador *senior* y "director de ingeniería" en una compañía tecnológica grande, que quiere dejar de serlo por el exceso de burocracia de dicha posición; o el de un *freelancer* con un historial de trabajo y entrega de proyectos ambiciosos y complejos. Los criterios principales son la capacidad de adaptarse a nuevos entornos y equipos, de combinar una buena disciplina de procesos con flexibilidad y, sobre todo, el amor por desarrollar cosas y ver a la gente usarlas.

A la hora de encontrar un CTO, los reclutadores probablemente serán útiles. Sin embargo, es importante informarles que estás abierto a personas con carreras poco convencionales y a

No lo desarrolles

"currículos irregulares". Una vez que encuentres candidatos, hay que preguntar cosas como:

- *¿Hay algún ejemplo de un proyecto que te sientas orgulloso de haber cancelado?*
- *¿Quién es el mejor compañero que has tenido, y cómo encontrarías más como ellos?*
- *¿Quién es el peor compañero que has tenido, y cómo evitarías a ese tipo de personas?*
- *¿Has conseguido asesorar a alguien que pasó del "peor" camino al "mejor" camino? ¿Has estado a punto de conseguirlo? ¿Qué hiciste?*
- *¿Cómo decides si una tecnología pertenece al stack de un proyecto?*
- *¿Cómo reaccionas cuando un proyecto se retrasa (mucho)? ¿O cuando nadie lo está usando?*

No hay respuestas perfectas a estas preguntas. Lo que se busca es alguien que haya sido capaz de evitar que su equipo cometiera un error, o desperdiciara un gran esfuerzo; que sepa cómo formar un equipo, tanto en sentido positivo como negativo; que tenga ideas de cómo guiar a los miembros jóvenes del equipo; que esté en constante sintonía con las nuevas tecnologías, pero también con los aspectos prácticos de su aplicación sobre el campo; y que responda bien cuando algo salga mal. Si encuentras a alguien que conozca sobre el contexto en el que van a operar, consideralo prioritariamente, aunque no de forma definitiva — es mucho más probable que funcione bien alguien que sólo tenga una familiaridad limitada con el contexto pero que es un excelente mentor con una gran capacidad de adaptarse y equilibrar el rigor y la creatividad, que alguien que conozca el contexto a profundidad pero no pueda adaptarse o ser un buen mentor.

Por último, hay algunas banderas rojas. Evita a las personas que ya no están aprendiendo. Evita a las personas que se autopromocionan. Ten cuidado con alguien que diga algo como: "No me veo escribiendo mucho código, porque creo más valor formando y administrando al equipo que escribiendo código". Por supuesto, si el equipo crece de una manera significante, escribir código ya no será el mejor uso del tiempo del CTO. Pero el tipo de director de tecnología que trabaja bien en equipos pequeños probablemente tendrá que escribir mucho código por sí mismo. Si no disfrutan de esta actividad, si quieren administrar más que desarrollar, es poco probable que funcione en tu equipo.

pagas más por el código que no se escribe, que por el que se escribe

Contratistas senior

El mejor contratista *senior* con el que he trabajado me dijo una vez: "Me pagas más por el código que no escribo, que por el que escribo". Era una exageración, pero resume una característica fundamental del valor que aporta un ingeniero *senior* — la experiencia para saber cuándo un camino que parece prometedor puede acabar en desastre, o que no vale la pena comparado con otra ruta más corta a un resultado similar que requiere solo una fracción del tiempo y energía.

La ejecución de un proyecto tecnológico es una cadena de compromisos. Los buenos contratistas *senior* ayudan a mejorar los resultados de estas decisiones, ampliando el rango de posibles opciones y buscando formas menos complejas de conseguir resultados (o teniendo confianza en que no existe ningún camino menos complicado). Es difícil saber de antemano que tan bueno es un contratista en estos aspectos, pero se puede evaluar en las entrevistas, preguntando por ejemplo:

- **Describe una ocasión en la que hayas conseguido evitar que un equipo tomara una mala decisión.**
- **¿Cuál es la peor experiencia que has tenido en un proyecto? ¿Cuáles fueron las peores concesiones que se hicieron?**
- **Personalmente, ¿cómo sabes cuando te estas complicando demasiado? ¿O cuando lo estás haciendo demasiado simple?**

Una bandera roja aquí es si un contratista expresa una opinión dogmática contra un marco o tecnología popular. Si tienes empleados jóvenes y los contratistas *senior* tienen que ser mentores o transferir conocimientos, pregunta cómo han manejado situaciones similares anteriormente.

Para buscar contratistas *senior*, hay algunas plataformas muy buenas que se especializan en encontrar y filtrar talentos de alto nivel (por ejemplo, Toptal). Pueden cobrar precios muy altos, pero para periodos cortos los gastos son manejables. La calidad del filtrado de posibles empleados que realizan, además de su habilidad para entender y encontrar habilidades nicho hacen que valga la pena. Por supuesto, sus resultados dependen de un equipo técnico capaz de integrar y utilizar rápidamente a las personas con conocimiento especializado. Eso será imposible si los procesos son demasiado rígidos, el resto de tu equipo demasiado dogmático o, por supuesto, si no cuentan con un CTO quien supervise todo el proceso.

Diseñadores de UX/UI

Los buenos diseñadores de experiencia de usuario e interfaces deben ser capaces de adaptarse a cualquier contexto, y seguir el comportamiento de las y los usuarios a donde sea que los lleven. El reto al contratar diseñadores es que es muy difícil identificar los elementos que son suyos en un proyecto terminado. Cuando un proyecto está terminado, se han producido tantas iteraciones que los diseños originales han sido relegados desde hace mucho tiempo. Un producto que termina siendo muy diferente a la idea inicial puede indicar que la idea inicial era mala, o puede indicar que tienen un diseñador flexible y creativo que trabaja sin problemas con un equipo de desarrollo para tomar nuevas direcciones.

Probablemente no exista una solución real a este dilema. En las entrevistas, hay que preguntar sobre ejemplos de trabajos anteriores para entender cómo un diseñador aprende y se adapta. A un amigo mío le gusta analizar un solo *wireframe* o una pantalla y profundizar sobre ella. Observando esto, se puede ver hasta qué punto pensaron en el diseño, cuánta responsabilidad tuvieron realmente sobre él y que tan buenos son para entablar conversaciones sobre el proyecto. Si se va a contratar a un diseñador por un periodo de tiempo largo, es razonable asignarles una tarea de evaluación durante el proceso de contratación, siempre y cuando esta tarea sea corta y concisa. También puede ser una buena idea evitar a los diseñadores que son muy rígidos sobre sus métodos o dogmáticos sobre su técnica. Si alguien no puede explicar por qué los personajes modelo[2] que se tienen pensados son una mala idea a veces, puede que sea un buen candidato para una empresa grande y corporativa, pero probablemente no lo sea para una organización pequeña o para un equipo que opera en contextos imprecisos.

2 Los personajes modelo (o *personas*, en inglés) son personajes ficticios que se crean a partir de una investigación para representar a los diferentes tipos de usuarios que podrían utilizar su servicio, producto, sitio o marca de forma similar.

En general, cuando encuentres un buen diseñador, haz todo lo posible para que disfrute trabajar con ustedes. Teniendo en cuenta lo difícil que es contratar en este campo, y la gran diferencia que supone para un proyecto, no hay casi nada tan valioso en el ámbito de la tecnología del desarrollo como un diseñador excelente al que le guste trabajar contigo.

Ingenieros jóvenes

Las universidades, los *bootcamps* y los cursos en línea producen cada año un flujo constante de personas quienes han escuchado que la programación está en gran demanda, es bien pagada y proporciona una carrera satisfactoria y estimulante. Al poco tiempo de terminar, reciben una docena de rechazos laborales. Puede que lleguen a una “entrevista técnica”, donde se les hacen preguntas irrelevantes sobre cosas que nunca van a hacer, e inevitablemente son rechazados. Los que consiguen pasar por este absurdo filtro terminan recibiendo tareas insignificantes y a menudo sin sentido en proyectos enormes. Después, todo mundo se pregunta por qué cada año se produce software tan terrible y en tan grandes cantidades. Esta situación sería cómica si no fuera tan trágica. Sin embargo, es útil para quienes ocupan reunir un equipo de desarrollo con un bajo presupuesto. Significa que puede contratar a programadores jóvenes con gran talento, casi siempre procedentes del contexto en donde opera el proyecto, y darles una vía para superar el absurdo del nivel de entrada al mercado laboral donde les piden dos años de experiencia en su CV, ofreciéndoles un salario modesto. Puede que se vayan después de 18 a 24 meses, y tal vez necesitarán 6 meses para ponerse al día, pero en el periodo intermedio podrás construir a un ritmo y por un presupuesto que algunos (malos) no consideran posible.

Para hacer esto, tendrás que revisar muchos, muchos CVs, posiblemente cientos. No se puede automatizar ni subcontratar el proceso, o corres el riesgo de que se descarte al mejor candidato que se podría haber contratado porque alguien que

no le importó lo suficiente quería simplemente terminar de revisar los CVs. Así que tú y tu director de tecnología tendrán que pasar por ellos — puede llegar a ser el trabajo más valioso que puedas hacer. Si no tienes el tiempo y la energía para hacerlo, simplemente no desarrolles el proyecto.

Los criterios, tanto en los CV como en las entrevistas, que considero que han dado mejor resultado son:

- **Hambre de aprender:** *Algunas preguntas tradicionales — sobre proyectos externos, o temas aprendidos fuera del estudio/trabajo, o fracasos — son realmente útiles. También es útil preguntar por una decisión en equipo tomada en algún proyecto anterior, y las razones por las cuales se tomó: alguien a quien le gusta aprender en el trabajo habrá intentado entender la decisión y su justificación, mientras que otros se habrían limitado a respetar la decisión y seguir con su propio trabajo.*

- **Compromiso con la calidad:** *Programar implica tomar con frecuencia varios atajos y concesiones prácticas. Es una gran tentación tomar una salida fácil y justificar como tal, sin haber explorado adecuadamente todas las opciones. A menudo es muy difícil saber cuál es el caso, así que la gran tentación a la que se enfrentan los jóvenes programadores es utilizar la funcionalidad como excusa. Eso se puede comprobar con pruebas de código, pero solo hasta cierto punto. En este caso, las pruebas de código para llevar a casa son útiles, pero sólo si son discutidas a profundidad después de su entrega. También pueden ser útiles las preguntas sobre la resolución de problemas en sus experiencias anteriores y la formulación de respuestas hipotéticas a escenarios imaginarios (pero concretos y plausibles).*

- **Capacidad técnica básica:** *Suena obvio, pero (incluso aunque declaren tener experiencia) a veces es sorprendente la cantidad de candidatos que no saben programar. Por lo tanto, algunas preguntas sencillas de programación y algunas preguntas técnicas son útiles..*

El mejor programador que he contratado fue alguien quien primero intentó ser criador de pollos. El gobierno sudafricano le dio un subsidio para criar aves de corral, pero no le informó sobre la prevención de enfermedades. Mientras sus pollos morían, aprendió a programar. Cuando lo entrevisté, ya había creado un par de mini-aplicaciones para sus amigos y podía explicar de forma coherente las decisiones que había tomado al crearlas y cómo las modificaría en un futuro. Lo encontré a través de un reclutador igualmente joven, talentoso y poco ortodoxo, a quien le había dicho que buscará candidatos inusuales.

Definiciones:

Stack Overflow:
Un sitio donde los desarrolladores hacen y responden preguntas. Podría decirse que, después de Wikipedia, es el bien público más valioso de Internet.

PLAZOS Y PRESUPUESTOS

fija un presupuesto que te permita empezar rápidamente, pero con el que puedas seguir iterando a lo largo del tiempo

Es común que una versión sencilla de la tecnología que estás intentando desarrollar llegue a manos de usuarios reales (lo que se conoce como "lanzamiento") más rápido de lo que crees. También es común que el tiempo estimado para llevar la tecnología a un estado en el que se utilice constantemente y tenga pocos errores ("madurez") sea mucho más tiempo de lo estimado. Por este motivo, fija un presupuesto que te permita empezar rápidamente, pero con el que puedas seguir iterando a lo largo del tiempo, a la vez que conservas los conocimientos dentro del equipo.

En lo relacionado a software, la mayoría de las cosas salen mal. Tal vez nadie quiere usar lo que has desarrollado, o el producto falla de forma extraña e imprevista una vez que está en el mundo real, porque tanto el software como la vida son complicados. Tal vez el producto falla porque está mal elaborado. Siempre, siempre habrá errores, pero puedes disminuirlos considerablemente si **(1)** contratas a un buen director de tecnología, y **(2)** utilizas una práctica conocida como "DevOps", y (aunque esto es más controversial si no es un hábito del equipo) "desarrollo dirigido por pruebas" (*test-driven development, o TDD*).

Pero incluso la tecnología desarrollada con estándares altos va a encontrar defectos imprevistos una vez que esté en el mundo real. Esto puede alargar el plazo de construcción de la tecnología mucho más lejos de su fecha de lanzamiento. El software se compone de muchas piezas conectadas entre sí y se utiliza en un mundo muy complejo. Las y los usuarios se comportan de formas que no se entienden al principio y que no pueden ser previstas. Entre los comportamientos imprevistos y las complejas interacciones entre los componentes del software, puede ser muy, muy difícil identificar la razón por la que algo no funciona.

En resumen, los plazos de lanzamiento siempre pueden acortarse, y los de maduración siempre se alargarán. Para

No lo desarrolles

este tipo de guías, una regla común es nunca dar detalles específicos para evitar crear expectativas falsas. Otra regla común en el desarrollo de software es decir “se va a tardar lo que se tenga que tardar”. Voy a romper ambas reglas, y decir que para un buen equipo de 3-4, siguiendo las buenas prácticas, debes ser capaz de lanzar un producto moderadamente complejo en unos 3 meses, y obtener algún tipo de estabilidad y madurez en 12 meses. Después de esto, por supuesto, se vuelve a empezar — si tienes una audiencia. Si no la tienes, detente a los 6 meses, o mejor aún, a los 4 meses, o mejor aún — **no lo desarrolles.**

Lo que estas líneas de tiempo significan para los presupuestos varía según la estructura del equipo. En cuatro años, Grassroot gastó alrededor de $200,000 dólares en su desarrollo. Esto cubrió la creación de la plataforma principal, la ampliación para realizar campañas nacionales, experimentos con bots de WhatsApp y sistemas de aprendizaje automático, y varias aplicaciones para Android. Esta cifra está ligeramente distorsionada por el hecho de que la organización me tiene a mí como director ejecutivo y director de tecnología, con una tarifa mucho más baja que la de mercado, lo cual no recomiendo. Sin embargo, conozco casos de organizaciones que facturan mucho más que esta cantidad.

En términos monetarios, en los mercados en demanda, un buen director de tecnología (véase más abajo) tendría un salario de alrededor de 10,000 dólares al mes (y puede que sea más caro en las grandes ciudades de EE.UU.), y un buen ingeniero junior sin experiencia, alrededor de 4,000 dólares (y sería más caro en Nueva York/San Francisco). En los mercados en desarrollo, si no están distorsionados por el dinero de los “fondos de impacto” o de las compañías de capital riesgo, esas cifras serían de 5,000 dólares y de 1,000 a 2,000 dólares.

Para este tipo de guías una regla común es nunca dar detalles específicos para evitar crear expectativas falsas. Otra regla común en el desarrollo de software es decir “se va a tardar lo que se tenga que tardar”. Voy a romper ambas reglas

Definiciones:

Prueba:
Un pedazo de código que pide a tu software que realice ciertas funciones, y luego comprueba si se hacen correctamente.

Los programadores de Europa del Este *senior*, por ejemplo, tendrían un salario de entre 80 y 100 dólares por hora si se contratan a través de una buena plataforma, o entre 40 y 60 dólares por hora si se contratan directamente (casi siempre vale la pena contratar por medio de una plataforma, a menos que ya se haya trabajado con ellos o cuentes con una buena referencia.) En general, a menos que cuentes con experiencia técnica interna, debería costar entre 50 y 100 mil dólares (excluyendo el costo del director de tecnología, pues se repartirá entre muchos proyectos) desarrollar y llegar a la madurez de un proyecto moderadamente complejo. Si cuesta mucho menos, te estás engañando a ti mismo; si cuesta mucho más, deberías revisar tus cuentas.

Dos últimos puntos aquí, uno para el inicio de un proyecto y el otro para cuando se encuentra cerca de la madurez. Al principio, la inversión en *tooling* suele valer la pena — lo que significa no solo poner en marcha el canal de DevOps (ver Definiciones anteriores), sino también tener alguna documentación básica, convenciones de estilo de programación, guiones para poner en marcha a los nuevos desarrolladores y automatizar cosas como los informes de calidad del código y la exploración de vulnerabilidades. La inversión inicial en este tipo de herramientas aumentará la velocidad con la que el equipo puede trabajar en el futuro, no sólo en asuntos rutinarios, sino también cuando hay emergencias. El tiempo se recupera en un futuro, generalmente por mucho más del que se invirtió. Sin embargo, si alguien empieza a sugerir algo llamado *Kubernetes*, es una señal de que se están sobrepasando.

Cerca de la madurez, quizás la ley más antigua en el desarrollo de software es la Ley de Brooks: “Añadir mano de obra a un proyecto de software al final de un proyecto hace que se tarde más”. En general, si el proyecto parece estar a punto de convertirse en una catástrofe, añadir más presupuesto y personal puede ser la peor ruta que puedas tomar. La incorporación de más personal, la discusión de qué hacer con los nuevos fondos, el establecimiento de nuevos informes: todo ello consumirá el recurso más escaso — el tiempo del director de operaciones y del equipo *senior* — y lo desviará de la tarea más importante, que es diagnosticar el problema principal y solucionarlo. *Cuando* el proyecto haya cambiado de rumbo, y haya pasado de estar crónicamente retrasado y crónicamente descompuesto a avanzar rápidamente y de forma sólida, aunque con retraso, entonces los recursos adicionales pueden impulsar una aceleración.

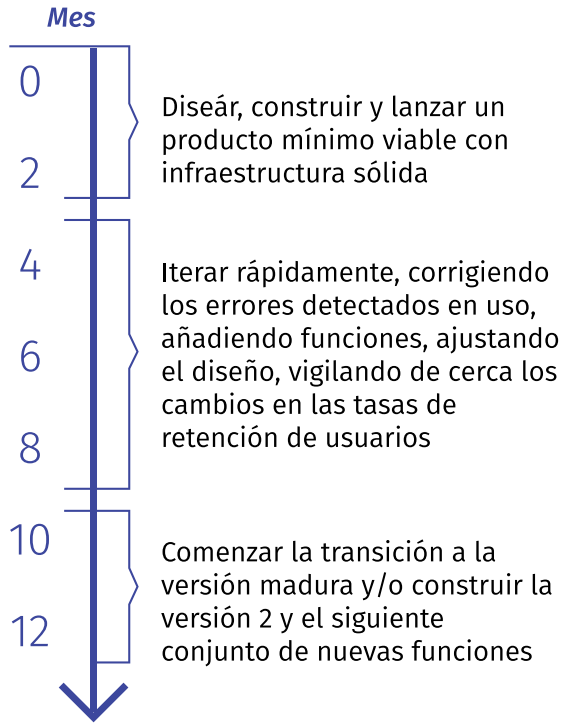
Definiciones:

Desarrollo dirigido por pruebas (TDD): Escribir las pruebas que el software debe pasar para ser aceptado, antes de escribir el propio software. TDD ahorra mucho tiempo a largo plazo y deja claro inmediatamente cuándo se ha modificado una función clave, a cambio de tiempo adicional por adelantado.

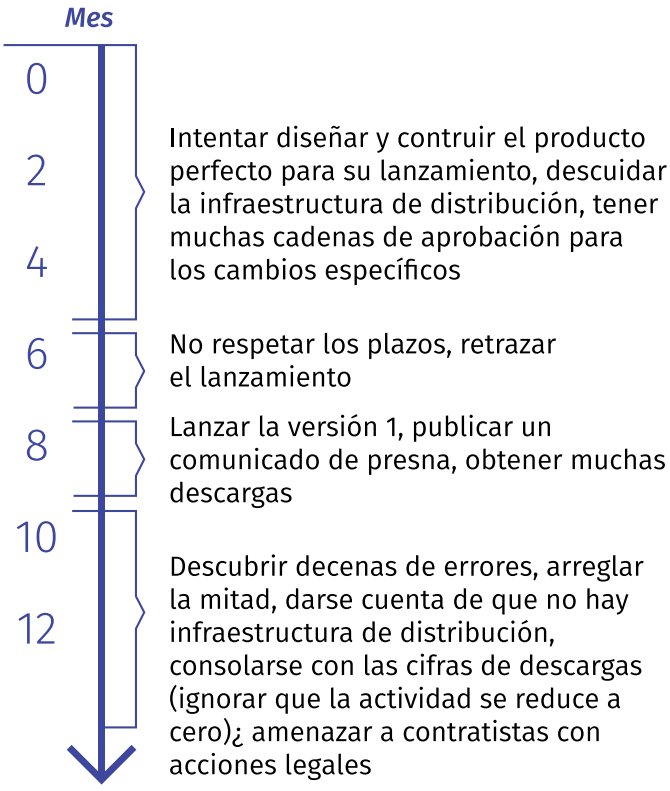
DevOps: conjunto de prácticas que permiten integrar continuamente el nuevo código en el existente y desplegarlo de forma rápida y confiable. Las herramientas específicas para integrar el código y desplegarlo se denominan “integración continua/despliegue continuo” (continuous delivery/continuous deployment, o CI/CD)

No lo desarrolles

Buen línea del tiempo



Línea del tiempo no ideal (planeada o actual)



Definiciones:

Script:
Un archivo que especifica una secuencia de comandos u operaciones a realizar, por ejemplo para crear una réplica en la computadora de un desarrollador de la aplicación que se está construyendo. A menudo se utiliza para automatizar ciertas tareas, como la configuración para alguien nuevo al proyecto, el archivo de registros o la comprobación de nuevas versiones de código.

Tooling:
Un programa que los desarrolladores utilizan para crear, documentar, limpiar, mantener, desplegar y dar soporte a otros programas y aplicaciones. Por ejemplo, un editor de código fuente o una herramienta para detectar y marcar posibles problemas de calidad del código

Kubernetes:
Un sistema complejo para gestionar el despliegue de aplicaciones a muy gran escala (millones de usuarios/as activos)

LAS Y LOS USUARIOS VS. LAS MÉTRICAS DE VANIDAD

La evidencia son los datos de la aplicación, recopilados de un número apropiado de usuarios, durante un periodo de tiempo adecuado.

Algunas de las tradiciones de Silicon Valley son útiles. La más útil es la idea de la "adecuación del producto al mercado" y su antítesis, las "métricas de vanidad". La adecuación del producto al mercado significa que se ha creado algo que la gente considera valioso y que utiliza con regularidad, y que se lo contará a otras personas. En otras palabras, ¿en verdad alguien quiere usar la tecnología que desarrollaste? Lo que importa aquí es que ni tu opinión sobre si la gente quiere usar tu producto — ni la opinión de tus socios, ni siquiera la opinión de tu focus group — son admisibles como evidencia. La evidencia son los datos de la aplicación, recopilados de un número apropiado de usuarios, durante un periodo de tiempo adecuado.

Es igualmente importante que las "personas que utilizan el producto" no signifique "total de descargas", o "páginas de aterrizaje", o incluso "cuentas creadas". "Usar" significa realizar la acción principal de forma regular y repetida, o — mejor aún — pagar (aunque sea con microtransacciones) por la tecnología. Las otras medidas son "métricas de vanidad", es decir, una métrica que presenta tu tecnología de forma positiva, pero que no es un buen reflejo de la realidad. El ejemplo más común es medir el número total de usuarios, que generalmente irá aumentando excepto cuando algo es un desastre total.

Las métricas de vanidad son especialmente peligrosas para los proyectos sin fines de lucro y gubernamentales, porque la prensa gratuita hace que sean fáciles de hinchar, y la falta de beneficios y pérdidas significa que tales métricas pueden interrumpir el aprendizaje indefinidamente. Además, los financiadores del proyecto y los directivos crearán incentivos fuertes, aunque implícitos, para utilizar las métricas que hacen que los objetivos sean más fáciles de superar, y obtener algo de prensa gratuita. Pero para un proyecto en sí, no hay nada más importante que un aprendizaje temprano rápido y

No lo desarrolles

la dependencia en métricas de vanidad interrumpe ese aprendizaje. Si es posible, *no envíes un comunicado de prensa* hasta que estés seguro de que el producto se ajusta al mercado y, aunque necesites las métricas de vanidad para tranquilizar a tus inversionistas, prohíbe su uso en las discusiones internas del equipo. Las métricas de vanidad son extremadamente tentadoras, sobre todo en periodos difíciles. Basta con que un miembro del equipo diga: "!No estamos tan mal, el total de usuarios sigue siendo muy alto!", para que la rigurosa honestidad de la que depende el progreso se detenga.

En su lugar construye un *funnel*, o embudo, desde las descargas y pasando por el registro, hasta el uso regular o la suscripción, y observa cómo los nuevos usuarios progresan a través de él. Para hacerlo, necesitarás datos detallados de tu aplicación sobre cómo la utilizan las y los usuarios, obviamente anonimizados y agregados. Esto se llama "instrumentación", y te permitirá observar cómo cambia el comportamiento medio entre grupos de usuarios (cohortes). Herramientas como *Amplitude* facilitan la configuración —no es necesario que la construyas tú mismo— y permiten a los miembros no técnicos del equipo supervisar la evolución del uso. Puedes configurar un gráfico que muestre el porcentaje de personas que abandonan el uso en cada etapa, y consultar este gráfico todos los días. Si no mejora, hay cosas por arreglar, independientemente de cómo la última campaña de las redes sociales haya presentado las métricas de vanidad. Si la retención de cohortes no aumenta, el proyecto no se ajusta al uso previsto.

En ese caso, ¿qué puedes hacer? Un conjunto de técnicas de experiencia de usuario y diseño de interfaz de usuario (UX/UI) puede ayudarte a entender en qué aspectos tu producto está fallando (preguntarle a las y los usuarios no es una buena idea: serán demasiado amables y puede que no se acuerden). Si no tienen un especialista en UX/UI, o hay poco presupuesto, aprende sobre las técnicas (espero que mis amigos de UX/UI me perdonen por sugerir esto) — a menos

No hay nada más importante que un aprendizaje temprano rápido y la dependencia en métricas de vanidad interrumpe ese aprendizaje.

Funne (Embudo)l:
A sequence of steps Secuencia de pasos por los que un usuario/a potencial transita en camino para convertirse en un usuario regular, desde la visita al sitio web o descarga del app (parte superior del embudo) hasta la configuración completa y el uso regular (parte inferior del embudo).

Definiciones:

Cohortes:
Un grupo de usuarios que comenzaron a utilizar el producto alrededor del mismo tiempo. Se utiliza para analizar si el ajuste producto-mercado está mejorando o empeorando.

Retención de cohortes:
La proporción de un cohorte la cual sigue utilizando el producto en una fecha posterior.

Tasas de conversión:
El porcentaje de usuarios en un determinado paso del embudo que pasan al siguiente nivel, por ejemplo, el porcentaje de personas que descargan el app o visitan la página web y crean una cuenta.

Instrumentación:
Herramientas integradas en un producto de software para medir en tiempo real cómo se está utilizando y cuándo y dónde está fallando.

UX/UI:
La profesión de diseñar experiencias de usuario (informalmente, el "flujo" de un producto) y luego convertir esa experiencia en una interfaz de usuario (el diseño específico).

de estar en graves problemas, en cuyo caso encuentra el presupuesto o recurre a algún voluntario con experiencia en UX/UI. Si se invierte en buen tooling e instrumentación por adelantado, debería ser fácil desplegar pequeños ajustes muy rápidamente, y observar los resultados en tiempo real. Aún necesitarás algo de paciencia y flexibilidad, pero no demasiada.

Encontrar el equilibrio entre la perseverancia y la estupidez es una forma de arte. Es normal que las tasas de conversión de usuarios bajen con la misma frecuencia que suben en los primeros meses después del lanzamiento. Pero si no estás constantemente acercándote a la adecuación del producto al mercado después de 3 meses, regresa a la fase de planeación y encuentra la manera de ajustarte. Si a los 9 meses del primer lanzamiento sigues sin acercarte a tus metas, es probable que sea el momento de suspender el proyecto y liberar los recursos para otra cosa. Otra bandera roja es si, unos meses después del lanzamiento, la tasa de retención de cohortes baja semana a semana durante un largo periodo de tiempo y los repetidos intentos (más de 3) de solucionar el problema no han tenido ningún resultado.

Una vez que tus cohortes mejoren constantemente y una buena proporción de tus usuarios se conviertan en usuarios regulares o de pago, la siguiente métrica a tener en cuenta es la *puntuación neta del promotor* (NPS, o *Net Promoter Score*). Esto mide la probabilidad de que la gente que usa tu producto se lo recomiende a otras personas. La manera más fácil de medirlo es incorporando un sistema de códigos de referencia en el producto y supervisando su uso, pero las encuestas ocasionales o automatizadas pueden servir como indicador.

La peor medida para saber cómo va tu producto es la cobertura de prensa. Debes desconfiar mucho de cualquier proyecto que tenga mucha cobertura de prensa. Es una señal de que el equipo está intentando obtener métricas de vanidad en lugar de hacer que su producto funcione como debe.

SELECCIÓN DE TECNOLOGÍA

Frecuentemente, amigos no técnicos que emprenden proyectos, o que simplemente quieren aprender a programar por sí mismos, me preguntan: "¿este lenguaje de programación es una buena idea?". En gran medida, el CTO que has contratado es quien debe tomar estas decisiones. Pero al igual que si diriges una organización de salud y no eres doctor es bueno saber lo suficiente sobre el tema para seguir las discusiones sobre técnicas médicas de alto nivel — es útil saber un poco sobre la selección de tecnología.

Lo más importante que debes saber es que, dentro de unos parámetros amplios, los detalles no importan demasiado. No dediques demasiado tiempo a qué lenguaje de programación usar o qué base de datos exacta, o qué usar para la aplicación móvil o la página web. Mejor utiliza un par de principios generales, y trata de evitar a quien intente convencerte demasiado de que su tecnología preferida es una solución perfecta — o bien están tratando de venderte algo, o no tienen experiencia con otros tipos de tecnología. Los principios básicos son:

(1) **Optar por el código abierto.** *En el caso de las organizaciones sin fines de lucro, esta elección viene impuesta en gran medida por los presupuestos. En el caso de los proyectos gubernamentales, las anécdotas sugieren que los proveedores empresariales están intentando vender agresivamente a los países en desarrollo para compensar el hecho de ser reemplazados cada vez más por el*

código abierto en el mundo desarrollado. No les hagas caso. Simplemente utiliza Postgres (o MySQL o MongoDB, si tu equipo tiene más experiencia con estos).

(2) **Elige un lenguaje popular.** *Como regla general, utiliza Python si el aprendizaje automático será importante, Java o C++ si tienen previstos millones de usuarios, y la familia de Javascript (NodeJS/Typescript) para otros casos. Pero mientras evites PHP (ya sé que dije que no hay que ser dogmático, pero PHP es realmente terrible), no importa realmente lo que uses. Elegir uno de los lenguajes más populares sólo significa que tienes un mayor número de candidatos para contratar, más frameworks para usar, y es probable que los problemas generales en los paquetes subyacentes se arreglen más rápidamente.*

Como mencionamos en varias secciones anteriores, debes exigir un desarrollo basado en pruebas. También debes asegurarte de que todo está instrumentado, es decir, que eres capaz de monitorear las estadísticas de uso en tiempo real y detectar caídas y errores, y, si estás usando contratistas, que la programación esté al día.

Lo más importante es que la elección de la tecnología no tiene casi ningún impacto en la seguridad. Una buena seguridad requiere disciplina en los procesos y una buena dosis de paranoia. Si alguien cree que su elección de una tecnología en particular va a proporcionarle seguridad, será hackeado, a menos que el producto no sea popular, en cuyo caso nadie se molestara por hackearlo. La seguridad empieza por el diseño. Una práctica conocida como *seguridad por diseño* va a ser central para la creación del sistema general, siempre y cuando sea práctico, para prevenir y contener ataques al sistema. Por ejemplo, las tablas que contengan cualquier dato personal identificable deben mantenerse en un servidor distinto al que se utiliza para procesar la actividad normal de los usuarios, y las funciones que se encargan de dicha actividad nunca deben tener permiso para acceder al otro servidor. Esta práctica coincide con un principio general de la buena ingeniería de software conocido como *separación de intereses*. Del mismo modo, hay que asegurarse de que se ha implementado el sistema DevOps (ver el apartado de Plazos y Presupuestos) y de que se han incorporado análisis de seguridad al proceso DevOps, de forma que el código se esté analizando automáticamente para buscar vulnerabilidades. Un código limpio es un código bueno, y un código bueno es un código (relativamente) seguro; los buenos equipos producen un código bueno e incorporan prácticas seguras.

Sin embargo, hay que asumir que, en algún momento, van a hackear tu proyecto. Si cuentas con un presupuesto grande, puedes pedir pruebas de penetración. Sin importar si puedes pagar esto o no, es importante mantener sesiones periódicas con el equipo de desarrollo para preguntar: si alguien nos hackea hoy, ¿cuál es el máximo de datos que puede obtener,

No lo desarrolles

dependiendo de cómo entre? ¿Cómo lo sabríamos? ¿Qué ha cambiado que podría abrir nuevas vulnerabilidades? El director de tecnología debe liderar estas discusiones. Pero los líderes no técnicos también deben participar — si no en las discusiones técnicas más específicas, al menos en el nivel de diseño del sistema y en la comprensión del potencial y los maneras para contener las brechas.

Por último, cabe señalar que si has decidido subcontratar todo el desarrollo de tu tecnología, a pesar de ser una pésima estrategia, puedes incluir todo lo anterior como requisitos en el alcance del trabajo. Tu director de tecnología tendrá que comprobar que el contratista cumpla con ellos. Si has subcontratado todo el desarrollo y no has contratado a un CTO, incluye los requisitos y espera que tengas suerte y el contratista te preste atención.

Definiciones:

API:
Interfaz de programación de aplicaciones. Una forma de que una aplicación (por ejemplo, un app en tu teléfono) se comunique con otra (por ejemplo, la función basada en la nube que realiza el procesamiento).

Framework:
Una base de paquetes de código pre-construidos que permiten a los desarrolladores crear nuevas aplicaciones rápidamente. Por ejemplo, React es un marco para el desarrollo web.

Prueba de penetración:
Cuando un hacker contratado o amistoso (a veces llamado “sombrero blanco”) intenta penetrar en su sistema para revelar vulnerabilidades para que puedan ser corregidas.

CONCLUSIÓN:
LA REALIDAD

A pesar de las mejores prácticas y las mejores intenciones, vivimos en el mundo real. Es más o menos inevitable que las fuerzas de la atención de la prensa, las preferencias de los inversionistas, las preferencias de los equipos y la inercia institucional hagan que te decidas a construir alguna tecnología por razones inadecuadas. También es posible que te presionen para que hagas "algo digital" como parte de la "transformación digital", o algo por el estilo. Las métricas de vanidad van a surgir. Alguien recomendará cobertura en la prensa. Antes de que te des cuenta, estarás conectando organizaciones no representativas con instituciones irresponsables a través de *chatbots* inservibles, justificando esto con la existencia de unos cuantos miles de personas que probaron el producto y no volvieron a utilizarlo. O bien puede que encuentres un proyecto que realmente valga la pena, pero no podrás contratar a un buen jefe de tecnología a tiempo completo, por lo que acabarás con un contratista desinteresado a tiempo parcial, y unos cuantos ingenieros junior sin mentores, y un proyecto que se lanza con retraso y nunca es arreglado.

En la actualidad, muchos financiamientos y expectativas no se estructuran para el éxito de los productos tecnológicos. En Grassroot, tuve que dedicar mucho tiempo a averiguar cómo hacer que una estructura de equipo flexible y adaptable funcionara con financiamientos de patrocinadores que hacían distinciones entre salarios de tiempo completo y parciales, incluso cuando se financiaban actividades esenciales. La idea de que un desarrollador junior se vaya, que un contratista intervenga brevemente, que se descubra un problema de usuario totalmente nuevo, que se incorpore otro contratista y situaciones similares no es un modelo que les resulte familiar a la mayoría de los inversionistas. De la misma manera, cuando nos dimos cuenta de que incluso nuestras métricas más profundas de actividad y participación de los usuarios — las

No lo desarrolles

cuales eran inusualmente altas — no llevaban a un impacto, algunos inversionistas seguían esperando aumentar las cifras de vanidad al expandirse a nuevas ubicaciones. Cuando encontrábamos proyectos con buenos resultados pero con métricas de vanidad bajas, no encontrábamos inversionistas interesados. Y muchas veces amigos o compañeros me pedían consejos y yo sugería no desarrollarlo, pues lo que fuera se desarrollaría de todos modos, obtendría un comunicado de prensa y quedaría en el olvido.

Sólo hay algunos caminos que puedes seguir desde este punto. Espero que los argumentos de esta guía te ayuden a evitar malos proyectos, estructurar bien a tu equipo y a aprender, crecer y lanzar productos más rápido. Si es así, la guía habrá servido para algo. Finalmente, tengo un par de consejos basados en mi experiencia. El primero es, en la medida de lo posible, cultivar un grupo de buenos ingenieros, diseñadores de confianza y líderes de equipo, y contar con ellos siempre que sea posible. Si tienes que hacer un proyecto y no puedes estructurarlo bien, pero tienes gente buena y de confianza que no está disponible de inmediato, entonces retrasa el proyecto hasta que estén disponibles. Conecta el equipo de campo y el de desarrollo continuamente. Solicita más tiempo entre informes porque “tenemos que utilizar la tecnología X, y la única persona que puede hacer este proyecto con la tecnología X no está disponible todavía”, o simplemente incorpora el tiempo al presupuesto.

Después, cuando surja un proyecto, dedica mucho tiempo con tus colaboradores de confianza a desarrollar una estrategia para obtener la máxima

Acercarse a las personas que usarán la tecnología y hacerlo rápidamente. Acercarte a tu audiencia significa contar con un equipo dedicado a la participación de la comunidad, estar presente sobre en el campo en la medida de lo posible

flexibilidad estratégica posible y la capacidad de ejecutar bien el proyecto, dentro de las limitaciones que la realidad impone. Si eres consciente y prudente al respecto, y has acumulado cierta experiencia, a veces es sorprendente el espacio que puedes crear.

Por último, la herramienta más importante es acercarse a las personas que usarán la tecnología y hacerlo rápidamente. Acercarte a tu audiencia significa contar con un equipo dedicado a la participación de la comunidad, estar presente sobre en el campo en la medida de lo posible y alimentar el proceso de desarrollo con observaciones directas en todo momento. Cuanto más rápido lo hagas, más rápido podrás iterar hasta conseguir algo que la gente quiera usar, y más rápido podrás ajustar una mala idea para que sea al menos parcialmente útil.

En resumen:

¿Puedes evitarlo?

No lo desarrolles

si tienes que hacerlo:

- contrata a un director de tecnología
- lanza temprano
- madura a través del tiempo

pase lo que pase:

- recurre a un equipo de confianza
- desarrolla de forma rápida sencilla
- acércate a las personas que usarán la tecnología y trabaja con ellos lo antes posible.

Más allá de eso,

como dice el Bhagavad Gita,

“tienes derecho al trabajo, no al fruto”

No lo desarrolles.

UNA GUÍA PARA PROFESIONALES DE LA TECNOLOGÍA CÍVICA / TECNOLOGÍA PARA EL DESARROLLO

GRASSROOT

MITGOV/LAB