Pregunta:

¿Algún comentario de carácter general? ¿Qué te parece el modelo, qué fortalezas y debilidades has identificado?

Respuesta (Manuel):

Conoces la página DevOps Topologies (https://teamtopologies.com/examples). Yo voy a dar un poco de feedback de vuestros trabajos en el sentido del valor para las empresas, que al final es lo que buscamos. En esta página empezamos a trabajar allá por 2014-2015 y fuimos incorporando cosas hasta que publicamos el libro Team Topologies, pero ya en esa página publicamos varios patrones y anti-patrones, y para cada uno se daba un contexto. Yo creo que realmente para las empresas esto es algo que les ayuda, porque no hay un modelo que te indique exactamente cómo tienen que ser los equipos o cuantos equipos de este o del otro tiene que haber.

En DevOps Topologies, lo que intentamos hacer fue ayudar a las empresas a pensar y reflexionar sobre si estaban más o menos cerca de un patrón o un anti-patrón y si en su contexto determinada estructura tiene sentido. Esto funciona para empresas con cierto tipo de características y tienes que estudiar muchas cosas como su tamaño, el tipo de producto que desarrollan, los procesos de contratación que hacen y el tipo de ingenieros que tienen, si externalizan muchas cosas (*outsourcing*) y hasta qué punto quieren tener control de ciertas cosas (por ejemplo, tener un gran departamento de operaciones o externalizarlo). En definitiva, hay muchos factores que pueden llevar una empresa a pensar: esto me parece una mala idea o una buena idea y en qué partes vamos a ir cambiando.

Bueno, el modelo UML tiene sentido, pero es algo muy estático. En mi experiencia las empresas necesitan **contexto** para poder sacar observaciones útiles. Yo creo que una taxonomía puede ser útil para ver qué tipos de organizaciones existen, pero luego hay que intentar dar un camino del punto A->B->C->D y esto es muy difícil puesto que hay mucho contexto que no podemos generalizar. En el UML lo que veo es que es muy estático porque estamos intentando fijar todas las variables. Por ejemplo, en Team Topologies nos enfocamos en que la gente entienda que esto es algo muy dinámico, tradicionalmente la organización de equipos es muy estática (estos son los equipos, los roles y el tipo de perfil que tenemos y ahora ejecutamos). Esto es lo tradicional, en dos pasos distintos definimos la organización y luego la ejecutamos. Y esto se va haciendo así cada X años y vamos reorganizando si algo no está funcionando bien o si hay una nueva tendencia (por ejemplo, ahora hay DevOps y tenemos que ser DevOps). Y es gran parte del problema, mirar esto como algo estático e intentar dar un gran salto, pero lo que tenemos hoy es una cosa y en el futuro tendremos otra y volveremos al inicio, pues tenemos otra organización.

En Team Topologies, por ejemplo, es muy importante la parte de interacciones entre equipos. Que no es solo colaboración, es colaboración, *facilitating*, y *X as a service*. Es verdad que en general, *X as a service* suelen ser los equipos de plataforma que ofrecen X a los equipos de producto, pero no es tan estático, puede haber diferencias.

Entonces la pregunta es cuál es el objetivo, qué queréis conseguir (aparte de publicar el paper). Si queréis ser una referencia que de una forma práctica las empresas pueden usar, no sé si el UML puede ser algo que ayuda pensar. Desde luego para ser una guía para las empresas tiene que ser algo más contextualizado. Ósea, el UML en sí la gente no lo va a mirar si le falta esto o

lo otro. Lo que tienes que hacer es mirar su situación y luego mirar las taxonomías que teníais.
Desde mi punto de vista los papers anteriores de cada uno me parecen más interesantes. Aunque ahí tengo algunas dudas en cuanto a las terminologías que usáis porque me parecieron un poco distintas de lo que yo escucho de las empresas. En definitiva, esas taxonomías me parecen más útiles. Se puede tener el UML como una referencia más interna y tener la taxonomía de los distintos tipos de equipos poniéndole un contexto encima, porque siempre hay que tener en cuenta el punto de partida de la empresa. Por ejemplo, si una empresa tiene mucho *blame* culture seguramente el departamento de desarrollo está contra el de operaciones, y a lo mejor los incentivos están mal porque esos incentivos están poniendo a los unos contra los otros. Ahí tienes estos tipos de problemas que tienes que buscar resolver, no hacer un cambio de equipos, porque si pones a gente de diferentes equipos que están acostumbrados a culpabilizarse juntos, la situación no tiene por qué mejorar. En conclusión, creo que para las empresas es más útil tener la taxonomía e indicarles los beneficios que van a conseguir y las precauciones que deben tomar para obtener esos beneficios (por ejemplo, si intentas hacer esto y no tienes una determinada cultura saldrá mal, o si no tienes la inversión necesaria no vas a conseguir que la automatización funcione).

Comentario (Dani):

Sí, yo creo que es cierto que la parte contextual es clave y tenemos que seguir trabajando en mejorarlo y crear una teoría sólida para la academia.

Respuesta (Manuel):

A nivel de teoría, yo creo que, bajo mi punto de vista, en este modelo falta la parte de **interacciones**, que también puedes modelar. Ósea, cuándo deberíamos tener una colaboración, o cuándo deberíamos tener una facilitación, o cuándo deberíamos tener *X as a service*. Y cuándo deberíamos volver de *X as a service* a colaboración. Yo estoy hablando con base en el libro
Team Topologies, aunque soy consciente de que no es la fuente de la verdad. Yo creo que el tema de las interacciones también se podría modelar, y creo que esto haría que la teoría ayudase a las empresas a evolucionar. De lo contrario, no sé si me vale de mucho la teoría porque simplemente estoy mapeando mi empresa con algo estático. Sería más útil si en el modelo
indicas primero tienes que colaborar, luego facilitar (porque en el modelo tienes *enabling teams*, pero luego no explicáis su funcionamiento), etc.

Comentario (Dani):

En nuestro modelo el *enabling team* se dedica a promover la cultura y los valores, y además hacen consultoría, facilitación y mentoría. Mientras, que los equipos horizontales en general son responsables de construir una plataforma y proporciona el servicio con esa plataforma a los equipos de producto. Y luego, para la **colaboración entre equipos** solo tenemos definida la frecuencia (alta, media, baja), no tenemos mucho más definido al respecto.

Respuesta (Manuel):

Vale, este es un punto que yo creo que falta y otro punto es a nivel de **plataforma**. Podemos decir que el tener equipos de plataforma es una taxonomía que suele funcionar bien, pero luego también hay muchos sitios en los que no funcionaría porque no han definido bien la plataforma.

La plataforma tiene que reducir la **carga cognitiva** de los equipos de producto que consumen la plataforma. Entonces ese es otro punto que no lo veo reflejado aquí. Porque esto no sirve para muchas empresas que tienen plataforma pero que, si la miras con detenimiento, esta plataforma es un conjunto de servicios un poco aleatorios, pero como son un servicio común, pues venga ponle el nombre de plataforma. Y entonces tienes un equipo de plataforma que está muy sobrecargado, porque tienes que hacer cosas muy distintas, tienes que hacer un servicio sobre CRM y luego tienes un servicio de infraestructura y son cosas que no tienen nada que ver. Simplemente porque el equipo de plataforma es un cajón de sastre.

Entonces esos dos puntos yo creo que se pueden modelar en el UML. Por ejemplo, para la colaboración puede haber una frecuencia, pero hay más cosas.

Pregunta (Dani):

¿Qué más parámetros crees que se pueden modelar sobre la

colaboración? Respuesta (Manuel):

Pues **duración**, **motivo** y una **validación** que cuando termina la colaboración y cuanto más específico mejor (si puede ser un booleano mejor). Por ejemplo, vamos a colaborar para definir esta API, pues el punto de llegada es tenemos la API definida y los dos equipos están de acuerdo en que ese es el punto de llegada. Otro ejemplo, necesitamos automatizar algún test, entonces el punto de llegada es tener un test automatizado en el pipeline. Entonces, para la colaboración son estos puntos: la duración estimada o esperada (que luego se puede cambiar) y el punto de validación que indica que se terminó la colaboración. Bueno, y también en algún momento puede pasar que la colaboración no esté funcionando porque uno de los equipos o no tiene tiempo o conocimiento para colaborar en esta área. Y luego la **facilitación** también debería tener un motivo, ósea porque estamos haciendo esto, e igual que antes saber cuándo se habrá terminado porque también hay un riesgo en la facilitación (para los *enablers teams* en particular, pero también puede darse facilitación entre equipos, por ejemplo, un equipo de producto senior que está ayudando un equipo más junior haciendo una migración de microservicios). Entonces, el tipo de interacciones puede estar más definido y de manera más específica y permitir que cualquier tipo de equipo interactúe con otro.

Incluso, si tienes el anti-patrón de silo (desarrollo y operaciones), puedes decir que en este momento Team Topologies (se refiere a un patrón DevOps) no va a funcionar porque nos pilla muy lejos, pero queremos empezar a tener una colaboración bien definida e intentar entender cuándo tiene sentido colaborar entre los equipos de desarrollo y operaciones. Pasa muchas veces que la colaboración entre equipos se convierte en una **dependencia**. Y bueno, puede ser, pero lo que pasa es que nunca llegas a quitar esa dependencia (por ejemplo, la colaboración con otro equipo que hace el despliegue u otro equipo de UX) y cada vez que tenemos un nuevo producto, tenemos que colaborar con ellos. Como resultado, lo que ocurre es que el equipo de producto no puede hacer las cosas por a sí mismo.

Pregunta (Dani):

¿Esos serían los *complicated-subsystem teams* en tu modelo?

Respuesta (Manuel):

No tanto, porque esos equipos son responsables de un servicio concreto, o una librería de

código o un servicio que otros equipos ejecutan. Deben tener *end-to-end*, el *ownership* de un servicio o una librería.
En el caso de UX, si hay un equipo centralizado de UX y muchos equipos dependen de este equipo centralizada, y esto es un problema (puede no serlo, pero normalmente lo es) porque dificulta el flujo de valor al cliente. Justo hace poco hable con una empresa que tienen ahí un problema porque hay un proceso que indica que el equipo de UX tiene que aprobar todo y eso les toma dos semanas y media para un cambio que ellos pueden hacer en dos días. Por eso **las interacciones sirven para reducir dependencias**, no para mantener interacciones con una determinada frecuencia (porque esto significa que tenemos una dependencia que a lo mejor no es útil a nivel de flujo), sino que muchas veces queremos colaborar para hacer una facilitación durante un tiempo y enseñar a un equipo, por ejemplo, a cómo hacer un buen diseño o cuales son las guías de diseño de la empresa, de forma que ese equipo pueda hacer, no el 100%, pero sí el 80% de las cosas de UX.

Pregunta (Isaque):

Bloque 1 (colaboración). Entendemos que la colaboración juega un papel primordial a la hora de organizar los equipos de un departamento de TI. ¿Qué factores promueven la colaboración o qué aspectos son impactados por una mayor o menor colaboración?

Respuesta (Manuel):

Justo lo que estaba comentando, yo veo la colaboración como un método para reducir dependencias. No es solo esto, muchas veces también sirve para que los equipos que tienen un problema común aprendan una solución. Por esto decía antes lo de los equipos de plataforma, hay que definir bien cuál es el propósito de este equipo porque tiene que colaborar mucho con los equipos producto. El equipo de plataforma ayuda a encontrar una manera de automatizar algo y de hacer algo de una manera más eficiente. Y también, muchas veces, el equipo de plataforma permite definir bien las fronteras entre equipos (por ejemplo, como se hace un servicio o cómo se gestiona la usabilidad, que también puede ser integrado en el trabajo de equipo de plataforma). Pero no puede ser que el equipo de plataforma decida todo solos y les diga a los demás qué tienen que hacer, ¡no! tiene que ser una colaboración.

Pregunta (Isaque):

¿Podemos observar esto como un contrato, entonces? ¿Con un objetivo y motivo para seguir la colaboración?

Respuesta (Manuel):

Sí, un contrato temporal digamos. La colaboración es importante, pero también tiene un coste alto, entonces vamos a usar la colaboración donde realmente tiene valor. No para un trabajo digamos más común en el que estamos colaborando porque hay una dependencia que no debería haber, una dependencia que deberíamos solucionar.

Pregunta (Isaque/Dani):

Bloque1. Pregunta1. Tenemos otra pregunta que está relacionando la cultura de compartir la responsabilidad y la propiedad del software. Si yo tengo un equipo que tiene propiedad compartida, ¿esto implica colaboración?, ¿el hecho de que un equipo de producto gestione bien

la propiedad compartida crees que promueve la colaboración?, ¿qué relación hay entre la propiedad compartida y la colaboración (interna y externa)?
Respuesta (Manuel):

Esta pregunta es interesante, nosotros en Team Topologies hemos visto que lo más eficiente no es solo la entrega, si no realmente entender el valor que se le da a los clientes y conseguir equipos autónomos. Un equipo autónomo tiene que compartir más responsabilidad dentro del equipo, incluso la definición de producto y la definición de valor al cliente.

Hay gente que dice que esto al final es un silo (que es lo que al principio decíamos que era un problema cuando empezamos con DevOps). Yo digo que sí es un silo, pero es un silo distinto que tiene valor. Con este tipo de silo a nivel de producto estamos diciendo que este equipo tiene un enfoque muy claro en un producto o en una parte de un producto. Y es verdad que este equipo tienen menos conocimiento de lo que pasa en otros equipos, pero esto lo hacemos aposta porque al final nos da más valor al cliente y reduce la colaboración (con otros equipos). Entonces, tenemos equipos más **autónomos**, con más **responsabilidad compartida** (end-to-end) y en los que esperamos ver menos colaboración con otros equipos (**menos dependencias**). Porque lo que estamos intentando hacer es un equipo más capaz de hacer todo el ciclo de vida completo. Pero también esto toma tiempo.

De ahí la importancia de las interacciones (colaboración, facilitación, y X as a service), que sirven para ayudar a estos equipos. Porque si no estamos hablando de equipos de menos de 10 personas que tienen conocimiento de muchas áreas distintas (infraestructura, desarrollo, arquitectura, diseño, etc.) y al final esto no es práctico. Bueno, vimos un caso en el que los equipos de producto casi no interactuaban con otros porque tenían todo esto, pero esto causa una carga cognitiva muy alta. Por esto, en Team Topologies proponemos que idealmente haya equipos de producto (que nosotros los llamamos stream-aligned Team) que tienen ownership end-to-end, y pueden hacer todo por ellos mismos y que tienen una misión clara (a lo mejor los *outcomes* definidos y qué quiere negocio ---incrementar ventas, por ejemplo---). En teoría, estos equipos pueden hacer todo por sí mismos, pero esto no es factible desde el punto de vista de carga cognitiva, de tamaño de equipos, etc. Además, muchos equipos no tienen el conocimiento necesario. Entonces, por esto viene Team Topologies y plantea la necesidad de equipos de *enabling* y un servicio de plataforma que tiene que ser construidos de manera que realmente reduzca, y no incremente, la carga cognitiva.

Pregunta (Isaque/Dani):

Bloque1. Pregunta2. Esto está un poco conectado con la próxima pregunta también. ¿En qué medida la colaboración puede contribuir a reducir los silos entre desarrollo y operaciones? Dicho de otra forma, ¿con qué frecuencia crees que tienen que colaborar para reducir los silos?

Respuesta (Manuel):

Tendrían que colaborar muy frecuentemente, porque esto significa que el equipo de desarrollo no sabe o nunca visto cómo se crea un servidor o un servicio en *cloud* (porque hasta ahora eso está dentro de un equipo de operaciones) y no tiene conocimiento de cómo hacer un *deployment* o cómo saber si un *deployment* tiene éxito o no y no sabe qué se está monitorizando. Todo esto está ocultado al equipo de desarrollo y no es visible para ellos. Entonces, sí, en este punto se necesita mucha colaboración.

Pregunta (Dani):
¿Y con qué frecuencia crees que se debe dar esa colaboración?, ¿diaria, semanal,

mensual? Respuesta (Manuel):

Más que con una determinada frecuencia, (diaria, semanal, mensual) es que la **colaboración sea dedicada y de calidad**. Es mejor colaborar una vez al mes, pero estamos una semana trabajando juntos y no estamos intentando hacer otras cosas. Porque luego, esto puede ser también un problema que tiene que ver con el *management* y la gestión de prioridades. El *management* tiene un papel importante en esto. Entendemos que la colaboración no es solo ejecutar el trabajo si no que es empezar a reducir los silos. Para eso tenemos que darle valor y permitir que los equipos estén 100% (que no es decir estoy aquí, pero esperando que me llamen para hacer otra cosa). Entonces, frecuente **una vez al mes por lo menos**, pero es mejor si es más frecuente claro, depende mucho del punto en el que estemos.

Por ejemplo, aquí tienes casos concretos. Tenemos que crear una infraestructura, una base de datos, por ejemplo, y para ello el equipo de desarrollo crea un ticket. El equipo de operaciones entonces va con ellos y lo hacen juntos. Este es un tipo de colaboración/facilitación, está entre medias. Cuando hacemos esto no es solo para crear la base de datos, sino para que el equipo de desarrollo entienda cómo funciona el proceso, qué tipo de herramientas se usan, etc.

Entonces, yo diría que la colaboración sea bastante frecuente al inicio, y luego pues habrá menos colaboración porque el equipo de desarrollo puede hacer ciertos *deployments* por ellos mismos o crear algún tipo de infraestructura. A lo largo del tiempo puedes notar que no tienes que colaborar semanalmente, si no mensualmente. Es una inversión realmente y más que definir si es diario, semanal o mensual, hay que definir la tendencia. Para esto también se necesita tiempo, pueden ser años para ver los cambios y que vaya disminuyendo la frecuencia.

Pero eso también luego tiene que ver con cambios de responsabilidades ya que, en las empresas, sobre todo en las que son muy grandes y también en las gubernamentales, suele haber mucha separación de responsabilidades y un perfil solo puede hacer un tipo de cosas. Pero eso es lo que tienes que cambiar también. A lo mejor hacen falta unos controles y desarrollo no puede hacer *deployment* directamente en producción, pero pueden hacer un *deployment e*n preproducción. Desde luego se pueden tener controles, pero hay que cambiar también responsabilidades y a veces los incentivos también.

Pregunta (Dani):

Bloque 2 (responsabilidad compartida): ¿Qué factores promueven que los equipos puedan fomentar la responsabilidad compartida? ¿En qué medida o qué aspectos son impactados por más o menos responsabilidad compartida? Es decir, ¿qué elementos tienen que tocar una empresa para promover la responsabilidad compartida en sus equipos?, ¿cómo pueden hacer que todos los miembros de un equipo sientan todo el proceso (desarrollo, despliegue, etc.) como suyo?

Respuesta (Manuel):

Hay dos puntos importantes dentro de esta pregunta: lo primero que estáis buscando es cómo promocionar el sentido de *ownership* y el segundo tiene que ver con la **motivación** intrínseca

de las personas (referencia al libro Drive de Daniel Pink).
Respecto a este último punto hay tres factores para la motivación intrínseca de las personas: tener un **propósito** con el cual uno se identifica, sentir que uno puede hacer bien su trabajo (**maestría**), y tener **autonomía** para tomar decisiones por uno mismo. Entonces, yo creo que estos factores se aplican también dentro de los equipos. Para que la gente se identifique con el propósito del equipo es preciso que esté bien definido, para lo cual hace falta mucha claridad en la organización acerca de los productos y lo que se ofrece a los clientes. Para que uno tenga autonomía tiene que haber una buena identificación de cómo descomponemos un producto, tanto a nivel de negocio como a nivel técnico. No vale simplemente con decir que eres un equipo de producto y puedes tomar tus propias decisiones si luego llegas y el código está compartido y no puedes cambiarlo, entonces no tienes independencia a nivel técnica para tomar decisiones. Muchas veces a nivel de negocio también hay problemas con la autonomía. Por ejemplo, si tienes tres personas pidiendo *features* que son conflictivas o prioridades conflictivas. El equipo no puede estar dependiente de muchas decisiones distintas y de distintos owners. Tiene que haber una cultura muy fuerte de que no es simplemente cada uno pida al equipo lo que quiere, si hay muchas cosas que hacer, los owners tienen que hablar entre ellos para decidir lo que quieren y lo que realmente necesita el negocio. En todo caso la autonomía está muy bien porque permite tomar decisiones de una manera más rápida y más eficiente dentro de un equipo. Y para el último, *mastery*, pues podemos pensar que el equipo aparte de tomar decisiones debe tener capacidad y poder aprender y mejorar su trabajo. Esto significa también no tener demasiadas responsabilidades. A veces vemos que los equipos grandes, naturalmente, tienen muchas responsabilidades y por ejemplo con los equipos de operaciones o de plataforma tienes que tener mucho cuidado. Porque si el propósito de la plataforma no está bien definido y son responsables de servicios muy distintos, luego no van a poder hacer un buen trabajo ninguno de ellos. En ese caso simplemente luchan por sobrevivir y mantener 10 servicios que no tienen relación entre ellos. Pero eso es otro tema, el de tener un propósito bien definido. Al final están los tres elementos relacionados.

Bien, ahora entramos en otro punto de la pregunta, hay que mirar no solo a nivel de equipos, si no más arriba e identificar bien los **value stream**. Podemos intentar tener equipos de productos que sean end-to-end y tengan responsabilidad compartida, pero luego también hay que saber cuáles son los value streams de negocio y saber qué es lo que ofrecemos a los clientes. Y saber también qué líneas de negocio distintas tenemos hasta que llegamos a un punto en el que se sabe cada value stream, los distintos productos y dentro de cada producto tenemos uno o más equipos que sean responsables de estos productos. En definitiva, tienes que entender el valor. Y esto lo hemos visto en algunas empresas que al inicio han empezado a adoptar Team Topologies en 2019 y se enfocaron mucho en alinear los equipos (stream-aligned, platform, enabling) y esto les ha aportado valor y han mejorado y tal. Sin embargo, han llegado a un punto en el que ya no lo hacen y es porque siguen con poca definición de los value-streams y ni se dan cuenta del problema. Realmente cuando ellos miran y piensan que no van tan rápido como esperaban, aunque los equipos tienen más cultura compartida, es porque no han hecho el trabajo de definir bien los values streams y alinear los values streams con los equipos para que puedan usar esta capacidad compartida para ir más rápido. Porque así siguen, vale, somos los dueños
del ciclo de vida, podemos desarrollar, diseñar, entregar y crear infraestructura. Podemos hacer todo nosotros, pero para saber qué tenemos que hacer, tenemos que hablar/colaborar con otros equipos, porque hay un proyecto que tiene que hacer un cambio y otro cambio que otro equipo tiene que hacer para una única *feature* o algo de valor. Entonces hay toda una parte de

identificar values streams y evolucionar los values streams que pueden faltar. Y hemos visto esto en
algunas empresas que terminan teniendo monolitos porque al final lo que han hecho es crear un producto pequeño que fue creciendo. Y esto se va haciendo de una forma un poco caótica, al final el monolito es técnico, pero es sobre todo a nivel de negocio ya que hemos incorporado muchas ofertas de valor distintas a clientes en un mismo producto y ahora está todo acoplado. Por ello lo que hay que hacer es entender los value stream.

Pregunta (Dani):

Pregunta 9. Entonces, ¿tú crees que para que un equipo sea stream-aligned (o equipo de producto) y sea end-to-end debe tener responsabilidad compartida obligatoriamente? ¿Tu dirías que es una condición imprescindible?

Respuesta (Manuel):

A ver, yo creo que es lo ideal. Y luego hay muchas posibilidades, por esto os decía antes que el contexto es tan importante. He visto solo un caso donde podemos decir que estos equipos tienen end-to-end completo (está descrito el caso en nuestra página, se llama U-Swich). Es curioso porque empezaron a hacer que cada equipo tenga end-to-end ownership y lo único que tenían que hacer de forma standard era usar los servicios cloud del amazon, nada más, no tenían
plataforma ni nada. Entonces, ahí los equipos eran stream aligned ideal en el sentido de no depender de ningún otro equipo, pero sin embargo llegaron a una carga cognitiva muy alta. Entonces tenían que preocuparse con tanta cosa, que ya ni siquiera tenían tiempo para analizar las necesidades de los clientes del servicio. Simplemente estaban ocupados con la parte técnica. Por esto, un equipo stream aligned por sí mismo no pueden sobrevivir, en el sentido que la carga cognitiva es muy alta y por esto necesitan que los equipos de plataforma y *enabling* les ayuden a reducir la carga cognitiva, para que puedan abstraerse de hacer ciertas cosas. Y muchas veces pensamos en la plataforma solo a nivel técnico más para servicios de infraestructura, monitorización, despliegue, etc. Pero, la plataforma puede hacer muchas más cosas, por ejemplo, plataformas de servicios de datos, productos internos que usan otros equipos, etc. Y esto siempre con el objetivo de reducir la carga cognitiva de los stream aligned.

En conclusión, si tengo un equipo de desarrollo, y queremos ir a stream aligned estoy aumentando la carga cognitiva de una forma muy grande pero los equipos stream aligned pueden aportar valor mucho más rápido y con más autonomía. Entonces, la clave es buscar maneras de reducir la carga cognitiva, como por ejemplo con las plataformas y los equipos de *enabling*.

Pregunta (Dani):

Ósea, que **en estos equipos cross-funcionales es inherente la responsabilidad compartida**, pero hay que reducir la carga cognitiva.

Respuesta (Manuel):

Si, además con estos equipos cross-funcionales surge en algunas empresas otro anti-patrón.

Ellos dicen ok, para ser cross-functional tenemos en el equipo un diseñador, un frontend, un backend, un tester, PO, etc. y al final necesitas un equipo de 15 personas y no tienen responsabilidad compartida porque cada uno está en su silo dentro de equipo. Esa idea de cross

functional puede llevar a este entendimiento que es un poco *naive*. Es mejor mantener un equipo

pequeño e ir a buscar cross-funcionalidad con más responsabilidad compartida que es típicamente el modelo T shape (cada persona tiene una especialidad, pero pueden hacer varias cosas distintas). De hecho, ahora hablan de un "Peine" shape, en el que tienes una especialidad, pero tienes que saber hacer muchas cosas diferentes.

Pregunta (Isaque):

Pregunta 11. Imagina que eres un stream aligned team, y aplicamos plataform servicing. ¿Podemos decir que estamos reduciendo la carga cognitiva de esa forma? ¿y al mismo tiempo es una forma de compartir la responsabilidad?

Respuesta (Manuel):

Sí, pero antes de contestar a la pregunta os quiero decir que en el modelo solo ponéis "*customer focus*" en los equipos de producto, pero **los de plataforma y enabling también tienen que tener *customer focus***, la única diferencia es que el customer es interno y no externo. Os digo esto porque tiene que ver con la pregunta.

Sí, en teoría **los servicios de plataforma reducen la carga cognitiva** si están bien hechos, si están ayudando los equipos de producto. Muchas veces tenemos buenas intenciones a nivel de plataforma, pero creamos un servicio que es difícil de usar/entender o no tienen los casos de uso que realmente necesita el equipo de producto, o no es viable porque muchas veces no está disponible. Por ejemplo, con CI/CD tenemos un sistema de plataforma interno, pero los viernes de 12 a 5 no está disponible y los equipos de producto no pueden hacer el trabajo de 12 a 5. Sí, el equipo puede hacer despliegue y cambios frecuentes, pero le estás complicando su flujo de trabajo. Entonces, todo esto no es un detalle, la clave para reducir la carga cognitiva es *customer focus* y esto implica colaborar con los equipos de producto para definir qué servicios/feature necesitan e incluso la plataforma no tiene por qué ser algo que se ejecute. Mira, incluso si nos dan una página wiki que nos explique cómo usar un servicio externo, esto para nosotros en Team Topologies, esta wiki es una plataforma en el sentido que se está reduciendo la carga cognitiva de los equipos de producto. Y pueden ser una buena manera para empresas pequeñas/startups que no tienen un equipo plataforma, pero que sí tienen algunas personas que se dedican a reducir la carga cognitiva. Por ejemplo: mira, usa estos tres servicios de amazon y usa esta configuración base, debería funcionar para la mayoría dos casos. Esto es un tipo de plataforma, que es un servicio que ayuda a reducir la carga cognitiva.

La otra parte es la responsabilidad. Yo diría que es importante clarificar que, aunque haya un servicio de plataforma para desplegar, la responsabilidad es del equipo de producto. El equipo de producto es el que tiene que hacer despliegue y usar la plataforma, pero no debería ser responsabilidad compartida de producto y plataforma. La responsabilidad es de producto, pues son los dueños del producto los que tienen que hacer el despliegue, hacer cambio y garantizar que funcione. El equipo de plataforma está enfocado en que sus servicios funcionan bien, no a que tu producto funcione bien.

Pregunta (Dani):

Bloque 3 (multidisciplinariedad). Saltando de bloque y hablando de la multidisciplinariedad. ¿Tu dirías que los equipos de plataforma también son cross-functional? ¿ósea todos tienen que saber hacer de todo?
Respuesta (Manuel):

Sí, de hecho, es un tema que sale muchas veces. Nuestra idea es que el equipo de plataforma es como un paraguas y dentro tienen los equipos que son los stream aligned. Pero estos equipos de plataforma dan un servicio o producto interno a los otros equipos, es un producto igual. Y deberían tener también, cross-functional skills, todos deberían saber hacer el diseño (aunque sea una API, tiene diseño también, puedes ser fácil o difícil de usar). Tienes que entender el caso de uso de los equipos de producto, todo esto es igual que un equipo stream aligned fuera o dentro de la plataforma. Incluso dentro de la plataforma puede tener otras topologias, el *enabling team* dentro de plataforma por ejemplo para ayudar a entender qué hace un producto, cómo desarrolla un producto, o puedes tener incluso un complicated subsystem team. Pero en general, son equipos stream-aligned y el otro tema es que tenemos plataformas que usan otras plataformas. Es como las muñecas rusas, dentro de un equipo de plataforma puede haber otro y tener distintos niveles de plataformas. Puedes tener a más bajo nivel infraestructura, operaciones, y luego tienes otros niveles como servicios de datos o incluso otras funcionalidades de negocio que realmente están en una plataforma porque son muy estables. Hay que tener cuidado, pero podemos tener *core funcionality* que está en una plataforma. Tenemos que tener cuidado, tiene que ser algo muy estable que no cambie mucho porque si no comenzamos a introducir dependencia.

Pregunta (Dani):

En general, ¿en una empresa en qué equipos crees que es clave promover la

cross-funcionality? Respuesta (Manuel):

Yo creo que en todos menos los enabling porque son un caso muy distinto. En mi experiencia, lo que es más duro es promocionarlo en los equipos de plataforma. Son los que están más lejos de esa mentalidad de que tenemos que tener varias capacidades para poder dar un servicio realmente ajustado a lo que necesitan nuestros equipos. Tienen un *background* más de infraestructura y operaciones clásico y nunca han trabajado en un producto. Están acostumbrados a recibir un ticket y ejecutar un script. Entonces, para ellos es muy distinto. Entonces, hace falta tener más ***product management* dentro de la plataforma** y es lo que a muchas empresas les falta. Los equipos de producto también muchas veces son equipos de desarrollo, aunque hacen más cosas y son *build-and-run teams*, muchas veces no tienen ownership de la parte de negocio y producto. Tienen un *backlog* que alguien define o su *product manager* define y el equipo ejecuta. Y aún falta ahí. Pero, el tema de cross-functional está bastante extendido entre los equipos de producto general y en plataforma no tanto.

Pregunta (Dani):

Bloque 4 (automatización). Vale, pasamos ahora al bloque de la automatización. ¿Qué aspectos o factores crees tú que promueven la automatización (desarrollar y desplegar sus productos de

la forma más automatizada posible) a nivel organizacional?

Respuesta (Manuel):

Yo lo que veo en la práctica, que la automatización está muy incentivada por las **herramientas** que hay. A nivel organizacional hay bastante distancia entre el entendimiento de la tecnología de la gente de negocio y la gente de la tecnología e ingeniería. Entonces va mucho porque la gente técnica dice esta herramienta es importante o deberíamos buscar una herramienta de CI/CD o
de *observability* que ahora está más de moda y entonces la gente de negocio se pregunta qué herramienta hace falta adquirir o si hace falta contratar consultores o lo que sea. Entonces, la automatización yo lo que veo es que avanza mucho por esto. No está mal, vamos mejorando, pero falta pensar en qué automatizaciones necesitamos para reducir la carga cognitiva de los equipos. Y también, se necesita automatizar a nivel de gobernanza y de temas como seguridad, privacidad, regulaciones. Todo esto tiene mucho sentido automatizarlo en algún tipo de servicios de plataforma, pero falta mucho para esto porque es mucho más difícil que comprar una herramienta. Este es tu negocio interno, y tú tienes que tener tus expertos que colaboran con plataforma para automatizar, no todo, hay cosas que no se puede automatizar, pero hay mucha cosa que puede automatizar (controles financieros, controles de privacidad, etc.) porque esto es otro tipo de dependencia que suele haber. Los expertos en seguridad, *compliance*, legal, esto es también un cuello de botella. Entonces es importante promover la automatización y aumentar así el flujo del valor y no tener todas estas dependencias. Sin embargo, hoy día no se piensa mucho en eso porque va más por las herramientas.

Entonces tendría que implicarse gente de negocio, que no tienen conocimientos técnicos tan profundos, y también de ingeniería claro. Los de negocio deben entender lo que se puede automatizar y el equipo de ingeniería tiene que ayudar a explicar lo que es posible. No es solo lo que me da la herramienta X, si no que nosotros (ingeniería) deberíamos estar mirando que tipo de problemas de negocio podemos reducir y automatizar con nuestro conocimiento.

Pregunta (Dani):

Pregunta 21. Claro, en línea justa con una cosa que has comentado sobre los equipos legales. ¿En qué medida crees que la existencia de los silos dificulta la automatización? Por ejemplo, ¿en qué medida el hecho de que haya un departamento legal o un departamento de UX que genere un cuello de botella dificulta la automatización?

Respuesta (Manuel):

Por un lado, podemos decir que facilita hasta cierto punto las automatizaciones porque cada silo puede decidir mejor lo que quiera automatizar. Pero, luego dificulta mucho porque en vez de automatizarlo cada silo lo intenta hacer dentro de su silo y esto les toman mucho tiempo o esfuerzo o intentan usar servicios o incluso *shadow projects* que no son oficiales y no son transparentes, lo cual es un problema de cultura ya que no pueden hablar abiertamente sus problemas. En general los silos dificultan. Ósea, no es 100% malo, porque hay casos que quieres automatizar de una manera concreta lo que necesita este equipo o este grupo de equipos porque es lo que ayuda, pero hay otros casos donde queremos compartir algunos flujos de automatización, datos y no tenemos las conversaciones para definir qué problemas son los específicos son para un equipo y qué problemas son comunes y deberíamos buscar una solución común.

Y en el otro extremo y de dónde vienen los *shadow projects* es que las empresas quieren todo standard. Entonces los equipos tienen las mismas herramientas, flujos, procesos. ¿Y qué te aporta esto? Vale, nos facilita el *reporting* y la visualización del sistema. Pero en el día a día de cada equipo los dificulta porque tienen que usar una herramienta que no hace falta. Pero vamos que en general los silos funcionales nos bloquean automatización porque o lo hacemos mal o no lo hacemos. Un ejemplo típico es que un equipo de desarrollo usa jira para gestionar su trabajo y el equipo de operaciones usa algo más específico como ServiceNow. El problema es que, si tienes un silo muy fuerte, no hay un tipo de conexión entre estas dos herramientas que están relacionadas. Por ejemplo, si estás haciendo un cambio en jira, al final que hacer un despliegue y entonces tienes que hacer un ticket en ServiceNow. Sin embargo, a la gente de operaciones ciertos datos de jira les interesaría ya que si hay un problema pues tienen que ver que ha cambiado en el código, quien hay pedido esto, etc.

Pregunta (Dani):

Bloque 5 (plataforma y *enabling teams*). ¿cuáles son las características principales de los equipos de plataforma y los *enabling team* y cómo estos impacta en la empresa?

Respuesta (Manuel):

Esto es muy buena pregunta, y también muchos me preguntan qué tipos de métricas deberían mirar estos equipos para ellos mismos.

Como definición, un equipo de **plataforma** es un equipo que tiene un objetivo primordial reducir la carga cognitiva de otros equipos y ayudarles a ser más autónomos, acelerando así su entrega de valor. Es muy importante la definición de reducir carga cognitiva, porque no es tanto el servicio o la herramienta que vas a crear o entregar, si no como lo vas a hacer. Es importante colaborar y validar que lo que hacen realmente ayuda a los equipos de producto y no es un trastorno que está incrementando la carga cognitiva. Y, además, debería ser un equipo cross-functional que entienda qué es lo que ofrecemos a nuestros equipos internos y tener la capacidad de evolucionarlo de una manera autónoma.

Los equipos ***enabling*** son equipos pequeños de gente con especializaciones fuerte en un dominio de información que trabajan también como mentores o profesores digamos de una manera muy activa con los equipos. No es un tradicional COE (*Center of Excellence*) donde la gente está preocupada más en *guidance*, en documentar y crear ejemplos y tal. Eso está bien, pero un equipo de *enabling* debería estar codo con codo con los equipos de producto ayudándoles en sus problemas específicos durante un buen periodo de tiempo. Ósea, cada equipo tiene su experiencia y sus problemas y la idea es que los equipos de *enabling* les ayuden de una manera muy concreta. Para ello, hay que definir la facilitación en el modelo, por ejemplo,
vamos a estar 2 semanas con vosotros haciendo un test automático en vuestra app. Y también lo que hace falta, desde un básico de un test, qué tipo de herramientas usar, cómo mantener el código y hacerlo juntos a través de *workshops*, *pair programming* o lo que haga falta ya que lo que estamos intentando hacer es enseñar. Entonces esto es una especie de consultoría interna, digamos.

Pregunta (Dani):

Pregunta 28. Entonces, con lo que los equipos de plataforma proporcionan no es realmente suficiente para lograr una gestión automatizada, ¿no?, ¿hace falta también esta especie de consultoría de los equipos de enabling?

Respuesta (Manuel):

Si, suele ser la combinación de plataforma y *enabling*.

Lo que es curioso es que en muchas empresas los equipos de plataforma también hacen la facilitación. Lo malo de esto es que es una responsabilidad más a los equipos de plataforma, y suelen estar sobrecargados. Si lo quieres hacer, yo lo que recomiendo siempre es definirlo muy bien, quién lo va a hacer, cuánto tiempo, etc. Y si una persona va a estar 2 semanas ayudando los equipos de producto a entender, por ejemplo, sobre *observability*, pues entonces esta persona no va a estar en soporte de operaciones, no va a estar en desarrollo de una nueva *feature*. Va a estar dedicada a la facilitación y a lo mejor si tiene tiempo libre puede hacer pequeñas mejoras internas.

Esto tiene que ver con otro tema importante: ¿cómo medir el valor de este tipo de equipos, sobre todo los *enabling*? Suele ser difícil de explicar hasta que la gente no ve en la práctica el valor. Entonces, yo con ciertos clientes que tienen reticencia y no saben cómo justificar su valor, pues les digo que su equipo de plataforma haga un poco de facilitación (*enabling*) para que empiecen
a ver cuál es el valor. Y luego que hable con los equipos de producto para saber si esto es útil o no para que vean en la práctica el valor y luego llegan a una conclusión de si necesitan un equipo de *enabling* o basta con los de plataforma. Pero muchas veces vendría bien tener equipos de *enabling* dedicados, pueden ser equipos pequeños con al menos 2 o 3 expertos, por ejemplo, en tema de SRE (también hay muchas variaciones). De hecho, la idea inicial de google es era un *enabling* que ayude a los equipos de producto a como trabajar en esos temas (SRE).

Otra cosa interesante que pasa es que se empieza a ver que en algunas empresas que la gente que va a los equipos de *enabling* son senior. Entonces es como un paso en su progresión. Vale, habéis probado que sois buenos ingenieros, que podéis hacer el trabajo muy bien, y ahora el valor está en ayudar los otros rápidamente mejorar. Entonces, tienen que ser un facilitador más que una persona que ejecuta. Esto es curioso, pero he visto un par de casos donde se ha hecho click y se ve esto como un paso más en tu progresión, no tienes que ser manager por ser senior.

Por otro lado, están las **métricas** para la plataforma. Yo lo que recomiendo es empezar con métricas de satisfacción de los clientes y adopción de la plataforma, es básicamente un indicador de reducir carga cognitiva. Y adopción, si creamos una plataforma y los equipos de plataforma pueden usarla o no. Si hemos hecho la plataforma y luego hay muchos equipos que no lo usan, pues puede que algo hayamos hecho mal.

Es verdad que sobre todo en el caso de los equipos de plataforma cuesta enseñar el valor a la gente de negocio porque es una inversión grande. Y el valor a nivel de negocio es que los equipos de producto van más rápido y pueden entregar valor más rápido, que es lo que los de negocio quieren. Pero ayudan a esto de una manera indirecta. He visto casos en los que queremos medir el valor de plataforma y se hace de forma un poco artificial. Por ejemplo, si el servicio de plataforma cuesta X y 10 equipos de producto lo están usando entonces el valor es

10X. Y esto es un poco artificial, lo que deberías mirar es si los equipos de producto están mejorando sus métricas (las métricas de DORA, por ejemplo) y si están contentos con la plataforma. Pues ahí hay algo, quizás no podemos decir que es una correlación directa, pero hay algo que está relacionado y son indicadores importantes de que la plataforma es útil. Al final si los equipos de producto van más rápido y mejoran sus flujos, la plataforma tiene valor. Pero sí que es algo difícil de que la gente de negocio entienda, porque lo ven como un coste que puede ser bastante alto, puesto que tienen herramientas de servicio de Amazon y otros, más los costes de los equipos. Entonces es un coste alto y lo que intentamos hacer es mirar en los equipos de producto la satisfacción con la plataforma, la mejoría en sus métricas y también enseñar otros casos de empresas que lo han hecho bien. Por ejemplo, Adidas, que hace unos años tenían casi todo externalizado y ahora tienen una plataforma interna muy potente.