unir
LA UNIVERSIDAD DE INTERNET

FORMACIÓN PROFESIONAL

UNIDAD FORMATIVA 8

DevOps en la organización

DevOps en la empresa

Índice

DevOps en la empresa

Los contenidos que se abordarán en esta última unidad formativa están relacionados con la aplicación de las tecnologías y técnicas que hemos ido introduciendo en las organizaciones.

En concreto, explicaremos a qué retos se enfrenta una organización que inicia o está en el medio de un proceso de **transformación DevOps**, proceso por el que una organización de un determinado tamaño decide parar máquinas, reconocer que no se está haciendo todo lo posible para sacar el máximo beneficio con los recursos disponibles y comienza a cambiar su manera de trabajar y de crear software.

Este proceso se detalla en la sección dedicada a la cultura DevOps, donde además hablamos de los **equipos multifuncionales** que se necesitan en este proceso, describimos los factores que **aceleran la adopción**, y vemos qué herramientas de software pueden servir de apoyo a estos procesos y ayudarnos en el complejo proceso de fomentar la colaboración entre equipos y derribar los silos de las organizaciones tradicionales.

La siguiente sección está dedicada a introducir las metodologías ágiles, disciplinas orientadas a organizar el trabajo de los equipos de una manera que siga los principios establecidos en el Agile Manifesto. Principios que están totalmente alineados con la transformación DevOps y son, por tanto, claves para esa transformación que estamos buscando.

Se explicará sobre todo en qué consiste **Scrum,** posiblemente el framework Agile más usado, y repasaremos los conceptos fundamentales del mismo como los sprints, posibles organizaciones del equipo y las 'ceremonias' que ayudan a organizar todo el trabajo, siempre primando la adaptabilidad, el feedback y la entrega continua.

Una vez repasado Scrum, lo confrontamos con otros dos marcos de trabajo Ágil muy conocidos como son **Kanban** y **Extreme Programming**, ambos diferentes de Scrum, pero también con similitudes que hacen interesante una reflexión sobre sus características y cuál podría ser el más adecuado en nuestro caso concreto.

La lección, y por tanto el curso, termina con una sección dedicada al concepto de la **adopción DevOps**, asociada a las herramientas y procesos que dan soporte a la transformación de una organización: en concreto, se describe el papel de los **equipos globales** en esa transformación, su labor y atribuciones fundamentales, y ponemos en valor el equipo de **DevRel** como motor fundamental de los procesos de adopción de DevOps en empresas grandes.

Objetivos

- Explicar los retos a los que se enfrentan las organizaciones que empiezan el camino de la transformación DevOps.
- Definir el impacto de la cultura DevOps.
- Introducción a las metodologías ágiles.
- Definición e importancia de los equipos globales en la adopción DevOps.

Cultura DevOps y su impacto en las organizaciones

Si quisiéramos escribir un decálogo para DevOps, probablemente uno de los elementos más importantes sería la colaboración. Recordemos que DevOps integra dos mundos, el del desarrollo y el de las operaciones. Estos individuos adquieren conocimientos y capacidades de ambos entornos, y se abren, por tanto, canales de comunicación dentro de las organizaciones que permiten alcanzar una mayor agilidad, una capacidad requerida para la Integración continua, la entrega continua y los frecuentes *releases* (lanzamientos).

En sus cimientos, DevOps es un movimiento cultural: involucra a las personas. Una organización puede adoptar los procesos más eficientes o tantas herramientas automatizadas como le sea posible, pero todas ellas serán inútiles si no están integradas con las personas que deben ejecutar esos procesos y utilizar esas herramientas. En la construcción de una cultura DevOps, por lo tanto, está la clave de la adopción DevOps.

Una cultura DevOps se caracteriza por un alto grado de **colaboración** a través de los roles, centrado en el bienestar y los objetivos del negocio (en lugar de los objetivos exclusivos de cada departamento), la confianza y el aprendizaje a través de la experiencia. La construcción de una cultura no es como la adopción de un proceso o una herramienta, ya que requiere de cierta ingeniería social de los equipos, y cada uno está compuesto por personas que tienen predisposición, experiencia y prejuicios propios. Esta diversidad puede hacer que la construcción de la cultura se convierta en un proceso extremadamente desafiante y difícil.

Identificación de los objetivos de negocio

La primera tarea en la creación de una cultura es conseguir que todos vayan en la misma dirección y trabajen por el mismo objetivo, es decir, la identificación de los objetivos de negocio comunes para el equipo y la organización en su conjunto. Cuando el equipo sabe cuál es el objetivo común por el que está trabajando y cómo se medirá el progreso hacia ese objetivo, disminuyen los riesgos y el miedo individual y colectivo.

DevOps no es el objetivo final de una organización, sino que es una herramienta que permite alcanzar los objetivos propuestos. Actualmente, DevOps debe abordar nuevos retos empresariales, por lo que la organización puede usar esos desafíos como punto de partida para identificar los objetivos que se desea alcanzar y entonces, desarrollar un conjunto de hitos o peldaños que permitan alcanzar esos objetivos y que los equipos utilicen como guía.

Creación de un ambiente de intercambio

Después de identificar los objetivos comunes de negocio, el siguiente paso en la construcción de una cultura DevOps es que los líderes de la organización trabajen con sus equipos para crear un entorno de colaboración e intercambio. Los líderes deben velar por la eliminación de todas las barreras autoimpuestas para que haya cooperación.

Los baremos clásicos por los que se suele valorar el desempeño de los equipos de operaciones son el ***uptime*** (tiempo de actividad sin caída) y la estabilidad; en los equipos de desarrollo, las nuevas características (*features*) entregadas y el tiempo medio para su realización.

La realidad es que esta práctica **enfrenta** a estos dos grupos:

- Operaciones: sabe que la mejor manera de protegerse es no aceptar ningún cambio.
- Desarrollo: tiene poco o ningún incentivo para centrarse en QA.

DevOps elimina esta fricción entre ambas áreas, valorando el éxito desde la concepción de una responsabilidad compartida en la entrega de nuevas capacidades y features de forma rápida y segura.

La esencia de la metodología DevOps es el trabajo **conjunto** entre los equipos de desarrollo y operaciones, a fin de crear una sinergia que se centre en la consecución de objetivos comunes. Para lograr este objetivo, las organizaciones necesitan fomentar una comunicación eficiente, permitir que ambos equipos compartan ideas y resuelvan problemas de forma conjunta. Esta 'rotura' de los silos permite a las empresas alinear a sus trabajadores, procesos y herramientas hacia un enfoque centrado en el cliente.

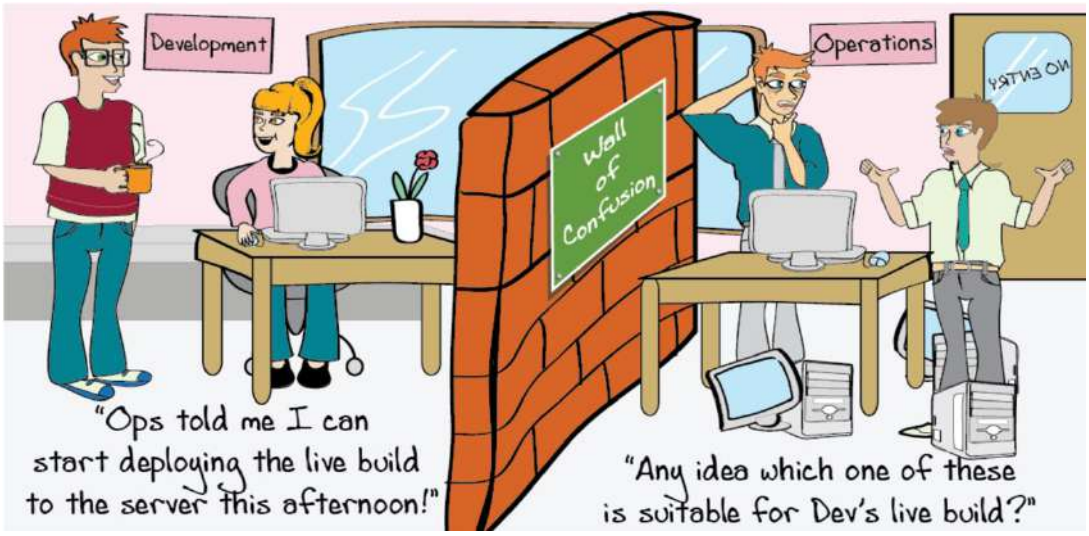

Figura 1. *What is DevOps?* Fuente: Buehring, Simon (2029). Knowledge Train UK. Recuperado de https://www.knowledgetrain.co.uk/it/devops/what-is-devops

Los líderes de la organización deben fomentar la colaboración mediante la mejora de la visibilidad. El establecimiento de un conjunto común de herramientas de colaboración es esencial, especialmente cuando los equipos están localizados en diferentes puntos del globo y no tienen la posibilidad de trabajar juntos en el mismo espacio físico.

Resulta crucial dar visibilidad a todas las partes involucradas en un proyecto sobre cuáles son los objetivos y el estado de un proyecto. Esto facilitará la construcción de una cultura DevOps basada en la confianza y la colaboración. Este cambio en la dinámica de grupos y el traspaso de la información tendrá un impacto a nivel humano y es probable que algunas personas deban ser reasignadas a otros proyectos mientras se esté adoptando esta nueva cultura.

Formar equipos multifuncionales

Es necesario que los equipos de DevOps se involucren en cada etapa del ciclo de vida del desarrollo de software, desde la planificación, la construcción, la implementación y la retroalimentación hasta la mejora continua. Para ello, es necesario que el equipo sea multifuncional y equilibrado, y que cada miembro posea un conjunto de habilidades idóneas para su rol en TI.

Además de estos principios existen otros que se centran más específicamente en el área de TI y que abogan por una mejora integral de sus procesos. Sin embargo, tienen un enfoque holístico para DevOps y organizaciones de todos los tamaños pueden adoptarlos. Estos principios son:

- Trabajar en entornos similares a producción durante el desarrollo y QA

El objetivo que persigue este principio es el de permitir que los equipos de desarrollo y calidad (QA) trabajen en entornos similares a producción a fin de que puedan verificar el comportamiento y desempeño de la aplicación mucho antes de que esté lista para su despliegue. Desde una perspectiva de operaciones, este principio tiene un enorme valor porque permite que el equipo pueda ver desde una fase temprana del ciclo cómo se comportará su entorno cuando soporte a la aplicación.

- Realizar despliegues mediante procesos repetibles y fiables

Esto permite que el área de operaciones de soporte a un proceso de desarrollo de software ágil e iterativo durante todas las fases del ciclo de vida que atraviesa el código.

La automatización es **esencial** para crear procesos que sean iterativos, frecuentes, repetibles y fiables, por lo que la organización debe crear una fuente o suministro de entregas que permita que el despliegue sea continuo, automatizado y validado a través de pruebas. Los despliegues frecuentes también permiten a los equipos poner a prueba sus propios procesos y a largo plazo hace que se reduzca el riesgo de fallos.

- Supervisar y validar la calidad operativa

Este tercer principio pone énfasis en la monitorización del código desde una fase temprana, al exigir que el código sea validado a través de pruebas automatizadas al comienzo del desarrollo y con una alta frecuencia, a fin de controlar las características funcionales y no funcionales de la aplicación. Esta supervisión frecuente proporciona una alerta temprana sobre cuestiones operativas o de calidad, que puedan ocurrir más tardíamente en producción.

- Favorecer y acelerar los bucles de retroalimentación (*fast feedback*)

Por último, el cuarto principio exige a las organizaciones la creación de canales de comunicación eficaces que permitan a todas las partes interesadas acceder y participar en un ciclo comunicativo. De ello, se desprende que el desarrollo de código tendrá el foco puesto en las prioridades del proyecto y podrá ajustarse a los requisitos del proyecto si estos merman. También implica que los pases a producción se harán mediante la mejora de los entornos de producción y que los planes de entrega se modificarán en pos del negocio.

Procesos DevOps

Anteriormente, hemos hablado sobre el papel que tienen las personas y la cultura en la adopción de DevOps. Los procesos definen lo que esas personas deben hacer y cuándo deben hacerlo. Por tanto, la organización puede tener una gran cultura de colaboración, pero si la gente está realizando las tareas de forma errónea o, por el contrario, están trabajando eficazmente, pero en el camino equivocado, el fracaso es todavía más probable.

DevOps es una capacidad que afecta a **toda la empresa**, haciendo que el negocio sea más ágil y que mejore la entrega de características a los clientes. También podemos ir más allá y entenderlo como un proceso de negocio: un conjunto de actividades o tareas que produce un resultado específico (producto o servicio) para los clientes.

El proceso de negocio de DevOps consiste en llevar las capacidades fijadas, normalmente por los directivos, dueños o inversores del negocio, a través del desarrollo y las pruebas, hasta que llegue a producción. Aunque este proceso de negocio no es lo suficientemente maduro para ser capturado en un flujo de procesos simples, es necesario poder capturar los procesos que la organización utiliza actualmente para producir y entregar las características. Esto servirá como punto de partida para realizar las modificaciones, cambios y vueltas de tuerca necesarias para impulsar el cambio hacia una cultura DevOps.

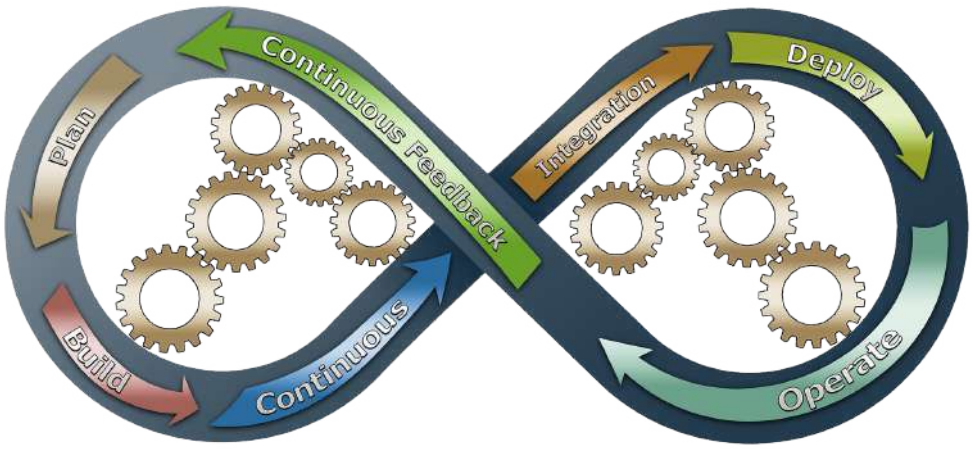

Figura 2. Ciclo Devops. Fuente: Desarrollo de Aplicaciones Multiplataforma.

En la figura podemos ver cómo los diferentes procesos DevOps forman un **ciclo continuo** que, a través del concepto del feedback, retroalimenta el proceso de planificación de nuevas características con información del proceso de operación y despliegue.

Factores que aceleran la adopción de DevOps

Para que la adopción de DevOps sea exitosa, es necesario que la organización tenga el objetivo de innovar. Sin embargo, también es frecuente que la decisión de implantar este enfoque se produzca por la aparición y repetición de problemas recurrentes.

Procesos pensados para controlar, no para habilitar

Imaginemos el siguiente escenario: un equipo de 100 desarrolladores necesita tener acceso a los recursos de pruebas para realizar sus tareas habituales y cotidianas (desarrollo, demos, etc.) y la empresa desea que estos recursos se consuman en la nube, con el objetivo de que sean más ágiles y evitar así una inversión en hardware.

En una organización tradicional podrían surgir muchos inconvenientes si se diese a los desarrolladores acceso a esos recursos: gasto descontrolado, poca visibilidad, imposibilidad de prever, etc. A continuación, probablemente se definiría un proceso para recontar y contabilizar los recursos se están consumiendo y, de alguna forma, aprobar este consumo. Por último, es posible que se generase un *workflow* (ciclo) de aprobaciones que, si bien funcionaría, dejaría mucho que desear en términos de eficiencia y agilidad.

Desde el enfoque DevOps debemos responder al interrogante de cómo se puede implementar un sistema para que los desarrolladores puedan acceder de forma eficiente y rápida a los recursos, con control y visibilidad. La respuesta, como es de esperar, será diferente a la de una empresa tradicional: se podrían definir políticas que permitan a los desarrolladores autoabastecerse de los recursos necesarios, mediante unas limitaciones definidas *a priori*.

Un ejemplo real puede ser el de una empresa que cuente con un portal autoservicio (*self-service*) con un presupuesto ya aprobado para permitir al equipo que invierta en herramientas y recursos de acuerdo con las políticas que ha dictado la empresa. De esta manera, se elimina la burocracia de los ciclos de aprobación. Recordemos que este enfoque favorece la independencia, la responsabilidad, la eficiencia y, por supuesto, la colaboración.

Falta de visibilidad

El hecho de establecer múltiples puntos de control es una forma de asegurar que las tareas, las personas, los recursos y demás elementos que componen la organización son gestionados de forma correcta. Sin embargo, también es sabido que todos estos puntos de control y procesos burocráticos pueden generar cuellos de botella o incluso generar una pérdida de tiempo y dinero por la monitorización de aspectos que no generan ningún beneficio a la compañía.

En línea con lo que hemos comentado en el apartado anterior, el hecho de tener una buena visibilidad reporta muchas más ventajas que ejercer un control excesivo. Por decirlo de otra forma, una buena visibilidad, en conjunto con altos niveles de responsabilidad y compromiso, permitirán al equipo reaccionar ante cualquier situación y cambiar de rumbo, mientras que un control férreo ralentizará a todo el equipo y no garantiza un resultado exitoso.

Otro caso de falta de visibilidad lo podemos encontrar en las organizaciones que utilizan cuentas AWS no asociadas, lo que eleva el gasto a nivel corporativo, porque al no haber una asociación de cuentas no se pueden aprovechar de los descuentos corporativos por volumen que ofrece el proveedor de nube.

Poca capacidad de innovación

La innovación en sí misma implica agilidad. Resulta imposible realizar cambios realmente significativos e innovadores en una organización encorsetada por una maraña de reglas. Debe haber un cambio de filosofía en las empresas que aporte más flexibilidad a sus procesos.

Algunas veces, es fácil identificar los puntos de mejora, sin embargo, resulta difícil encontrar la mejor solución posible a un determinado problema; aquí es donde la innovación entra en escena.

Hablando desde un punto de vista meramente técnico, además de un buen análisis de la situación y de las posibles soluciones, es importante que los equipos tengan la **libertad** para probar, cometer errores y aprender de ellos; en esto se basa principalmente la innovación, así que trataremos a este proceso como un proceso de:

- Definición.
- Prueba.
- Error.
- Éxito.
- Aprendizaje.

Un ciclo con tantas iteraciones como sean necesarias.

Dificultades técnicas evitables

DevOps aboga por la eliminación de la complejidad, ya que esta va en detrimento de la productividad y la eficiencia. Pues bien, en muchos entornos las dificultades técnicas que se perciben como 'técnicas' son en realidad dificultades humanas y a menudo están causadas porque se trabaja con procesos extremadamente complejos, que no hacen más que incrementar los puntos de fallo.

En las organizaciones tradicionales, existen ciclos de aprobación de cambios de código, comités de aprobación de pase a producción y, por supuesto, esto no hace más que añadir complicaciones a la hora de revertir el pase y corregir el código.

En una organización ágil la gobernanza debería estar lo más cerca posible del equipo DevOps, con el objetivo de que sean los responsables últimos de la implementación, verificación y puesta en producción de los cambios de software de los que son responsables.

Lamentablemente, en algunas organizaciones esto no es posible por razones legales: pensemos en un banco o una aseguradora, donde cualquier cambio por pequeño que sea debe pasar unas auditorías que garanticen que se cumplen los estrictos requerimientos que uno espera de entidades que manejan datos de alta sensibilidad.

Con todo, es nuestra labor como DevOps la de **facilitar** estos procesos, evitando en la medida de lo posible que se conviertan en un cuello de botella.

Miedo y resistencia al cambio

El miedo es endémico a la vida y al ser humano, de la misma forma que es el peor enemigo de la innovación y la colaboración. Nos encontraremos con el simple miedo al cambio, que incluso es un motivador positivo en las proporciones adecuadas, o el miedo a la pérdida de control, que en el caso de DevOps debería entenderse como un paso hacia la independencia y la visibilidad. Pero el peor, sin dudas, es el miedo a la pérdida del *status-quo* por parte de aquellas personas que quieren ser imprescindibles.

En un equipo DevOps, las personas pueden jugar diferentes roles en el transcurso de un proyecto, pero también comparten información sobre los avances, estados y alcance, entre otras cosas. Aquellos miembros que guardan información, conocimiento y herramientas creyendo que así serán imprescindibles, tienen cada vez menos cabida en las organizaciones que desean crecer.

Como indica Paz Cariñena Amigo (2016) las personas imprescindibles no atesoran los conocimientos como propios, sino que los comparten, están siempre predispuestas a ayudar a los demás y emprender nuevos retos. Estas personas son imprescindibles porque sin ellas las cosas se podrían hacer (no vamos a decir que no), pero con más esfuerzo o incluso con menos calidad. Pero lo más importante de todo, es que el mundo gira cuando no están, porque consiguen 'enseñar a pescar' y no solo dan pescado cuando se necesita.

Shadow IT

El shadow IT engloba a todas las tecnologías y elementos dentro de TI que están fuera del control de la organización o simplemente le son desconocidas.

Siguiendo nuestro enfoque de colaboración, innovación y agilidad, la falta de control no es el problema, sino que lo es la no estandarización y la falta de visibilidad derivada de ello. Es importante comprender que en muchos casos la necesidad imperiosa de ejercer control es lo que ha causado el shadow IT, es decir, la imposibilidad de acceder a los recursos en tiempo y forma.

Numerosas organizaciones grandes o medianas se han visto en la tesitura de estar usando recursos en la nube antes de poder siquiera diseñar una estrategia de nube.

Según Génesis Rivas (2018), entre los ejemplos más frecuentes de shadow IT podemos destacar el uso de:

- Aplicaciones de almacenamiento en la nube, como Google Docs o Dropbox.
- Redes no permitidas (como la red pública de internet).
- BYOD (*bring your own device*) no autorizado.
- Aplicaciones de terceros como las de sistema en la nube tipo software as a service (SaaS).

Una de las principales razones que motivan a los empleados a usar aplicaciones no autorizadas además de su bajo coste; es la facilidad que estas ofrecen en cuanto a acceso, mantenimiento y despliegue.

Entre las principales consecuencias que acarrea el shadow IT podemos encontrar (Rivas, 2018):

- **Fuga de datos sensibles**: a través de aplicaciones de terceros que no estén debidamente supervisadas por el equipo de seguridad de TI.
- **Procesos lentos**: el despliegue de ciertas aplicaciones puede ralentizar u obstaculizar el desempeño de los procesos de la organización.
- **Ineficiencia en la resolución de problemas**: en caso de que algún dispositivo o aplicación de shadow IT genere un incidente; puede resultar complicado para los profesionales de TI encontrar la fuente del problema.

Herramientas que favorecen la colaboración

Estas son algunas herramientas que favorecen a la colaboración:

Correo electrónico

Evidentemente, la principal manera de ponerse en contacto entre todo el equipo es el correo electrónico corporativo y lo lleva siendo desde hace años.

Existen diferentes maneras de obtener un servicio a nivel de compañía: podemos alojar y mantener nuestro propio servidor SMTP, con el coste humano que puede conllevar, o usar los servicios que nos ofrecen gigantes como Google o Microsoft (Gmail y Exchange), que tienen la ventaja, como veremos, de la integración con toda su suite de servicios.

En todo caso, en los equipos modernos el correo electrónico ha pasado de ser la herramienta fundamental de comunicación a ser una más, apoyada por otras como los servicios de comunicación instantánea más actuales.

Sistemas de comunicación instantánea

Las herramientas de comunicación deben servir para que los equipos puedan comunicarse entre sí de forma remota, enviarse archivos e información necesaria para el trabajo a distancia y organizar grupos con fines comunes, como por ejemplo 'Equipo Despliegues', 'Front-end React', 'Back-end Spring'…

Adicionalmente, muchos programas actuales ofrecen integraciones con otros servicios vía *webhooks*, de modo que se puedan tener mensajes automáticos en nuestros canales cuando termina una build de Jenkins o cuando una alerta de nuestra monitorización alcanza un determinado umbral. Esta característica resulta **fundamental** en equipos modernos de IT, por lo que no tenerla es un paso atrás que nos obliga a contar con otros medios para obtener esa información, como el correo electrónico.

Las principales herramientas que tienen estas características son Slack, Discord, Webex Teams y Microsoft Teams.

Slack, en concreto, es muy recomendable por ser una herramienta **muy potente** con posibilidad de uso gratuito, lo que la hace muy recomendable para equipos pequeños o grupos por fuera de la organización, como proyectos open source.

Herramientas de gestión de tareas

Se presentan a continuación una serie de herramientas que permiten gestionar las tareas de un equipo o una organización Agile y que además se adaptan perfectamente a los diferentes frameworks de desarrollo de software al permitir crear columnas con los nombres que deseemos, gestionar un backlog, editar las 'tarjetas' y añadir automatismos a las mismas.

Jira

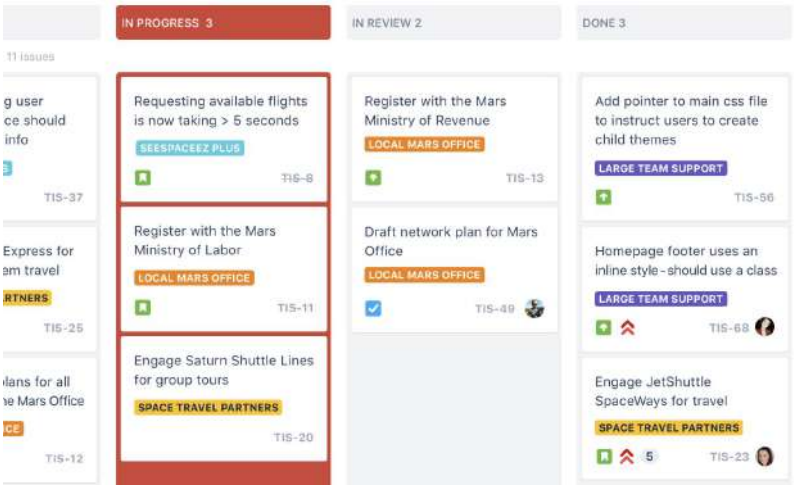

Figura 3. Fuente: https://www.atlassian.com/es/software/jira/features

Atlassian Jira es sin duda la aplicación de gestión de tareas más conocida, ya que además se adapta perfectamente a los frameworks de trabajo Agile más conocidos como Scrum o Kanban, permitiendo dar de alta todo tipo de tareas, gestionar un backlog y crear los diferentes sprints en los que se ejecutarán esas tareas.

También nos da la posibilidad de crear portales para alta de tareas personalizados, con lo que hace un servicio de *service desk* en el que las tareas que nos den de alta aparecerán como tareas JIRA más, que podremos gestionar en sus propios tablones.

Adicionalmente, tiene potentísimas utilidades para generar informes de todo tipo sobre las tareas que se han realizado: tiempo hasta finalización, *burndown charts*, velocidad del equipo, tareas recibidas frente a realizadas…

Por último, comentar que tiene disponibles un montón de plugins para integraciones de lo más diversas, como por ejemplo integración con Bitbucket (también de Atlassian), Github o Gitlab para ver los desarrollos asociados a un ticket o integración con sistemas de chat como Slack.

Trello

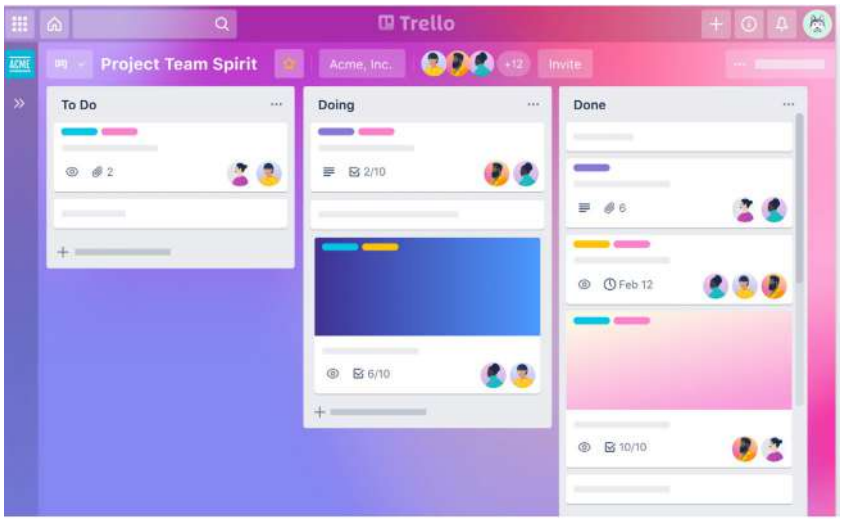

Figura 4. Fuente: www.trello.com

Trello es una herramienta gratuita, una alternativa a JIRA mucho menos potente, pero, sin embargo, con las funcionalidades suficientes para dar soporte a la gestión de tareas de cualquier equipo. Permite de forma **muy fácil** la creación de un tablero con un número cualquiera de columnas, asociar etiquetas a las mismas, acciones automáticas al mover las tareas…

Además, a través del sistema de los *power ups*, Trello nos ofrece funcionalidades extra de pago que lo hacen aún más potente (integraciones con Gmail, calendarios…) o desbloquean tableros extra que, en la versión gratuita, están limitados a un máximo de 10 tableros por persona.

Compartir ficheros

La necesidad de compartir ficheros entre miembros del equipo usando herramientas externas puede venir motivada por muchos factores. ¿Por qué usar una nueva aplicación, cuando podemos compartir a través de nuestros sistemas de mensajería instantánea, por correo o a través del control de versiones?

La respuesta es que las aplicaciones dedicadas ofrecen servicios extra que pueden resultar muy atractivos y que debemos conocer y valorar:

- Dropbox: proporciona muchísima capacidad de almacenamiento en la nube de forma gratuita, posibilidad de compartir carpetas y una carpeta física en nuestro sistema operativo que podemos usar para acceder a nuestro contenido en la nube como si fuese un recurso más de nuestro sistema.
    - Esto tan sencillo permite que tengamos una carpeta compartida en tiempo real con nuestros compañeros, por ejemplo. O compartida entre varios sistemas operativos, para poder trabajar tanto en casa como en la oficina.
- Google Drive es el sistema de almacenamiento masivo de Google, que obtenemos gratuitamente con cada cuenta de Gmail. Tiene la ventaja de que su uso vía web es muy sencillo y está totalmente integrado con toda la suite de Google, pero no permite su montaje como carpeta de sistema por lo que su uso directo es más limitado (no scripts).
- Microsoft One Drive es la respuesta de Microsoft a Google Drive, aunque tiene la ventaja de su integración total con sistemas Windows.

Suites integradas de productos

Existen una serie de SaaS por parte de proveedores que nos ofrecen paquetes completos de aplicaciones colaborativas, que incluyen documentos, presentaciones y hojas de cálculo compartidas, servidores de correo electrónico, sistemas de calendarios integrados…

Sin duda la gran ventaja de estos servicios es la integración que obtenemos *out of the box*, ya que están diseñados para aumentar la productividad de sus usuarios cuando lo único que usan son los servicios incluidos en su paquete.

Por contra, hay que considerar que se están metiendo todos los huevos en la misma cesta y quizás una migración en el futuro (por motivos de coste, de conectividad, de regulación…) sea muy costosa.

Pero desde luego son opciones muy potentes, que hacen muy importante considerar si el gran valor que aportan a nuestra organización compensa el coste de mantenimiento de la solución.

Office 365

Office 365 es un software as a service (SaaS) que se compone de una serie de productos y servicios. Todos los componentes de Office 365 se pueden gestionar y configurar a través de un portal; los usuarios pueden añadir contenidos manualmente o importarlos de un archivo CSV, por ejemplo. Además, puede configurarse un Active Directory local utilizando los servicios de federación de Active Directory para el inicio de sesión.

Entre las herramientas que incorpora podemos nombrar al servicio de correo electrónico, el gestor de tareas, la aplicación de calendario, el gestor de contactos y las aplicaciones de ofimática.

Una de las características más interesantes que presenta es su capacidad de funcionar en un entorno de nube donde todos los archivos residen en internet de forma segura. La lista de productos ofertados ha ido creciendo rápidamente, incluyendo Exchange, Skype para empresas (Skype for Business), SharePoint, Yammer, Flow OneNote, Dynamics 365 y la suite de Office Web Apps basada en navegador y el servicio Onedrive. Como novedad importante, existe la opción de acceso gratuito a las aplicaciones de Office Mobile para Android y dispositivos iOS (incluyendo tanto los teléfonos inteligentes y tabletas) para la edición y la creación de documentos. Por supuesto, otra de las grandes ventajas que tiene es que no es necesario preocuparse por las actualizaciones.

Google Docs

Google Docs es un software as a service (SaaS) y podríamos decir que es el equivalente de Google al Office 365 de Microsoft.

Permite crear documentos de texto, hojas de cálculo y presentaciones. Podemos utilizarlo también como un simple repositorio de documentos y archivos (Google Drive) o incluso como un editor web. Los documentos se guardan automáticamente en la nube de Google, y cuenta con una característica muy valiosa: permite ejercer un estricto control de versiones al dar al usuario la posibilidad de ver todas las modificaciones realizadas sobre el documento desde su existencia hasta la fecha.

Además, se pueden instalar diferentes plugins para hacer una edición de documentos en modo offline y que los ficheros se descarguen en el ordenador. Los documentos también se pueden exportar en varios formatos estándar como ODF, HTML, PDF, RTF, texto plano, Open Office, XML, etc. Por otra parte, los documentos pueden ser enriquecidos mediante etiquetas y metadatos. En lo referente a la compatibilidad, el servicio soporta todas las versiones recientes de Firefox, Internet Explorer, Safari y Chrome, que se ejecutan en sistemas operativos Windows, OSX, y Linux.

Google Docs sirve como una herramienta de colaboración para la edición de documentos en tiempo real. Los documentos pueden ser compartidos, abiertos y editados por varios usuarios **en tiempo real** y se pueden ver los cambios 'en vivo y en directo' (carácter a carácter) mientras los colaboradores sobrescriben y editan el documento.

Asimismo, Google Docs es compatible con los formatos de documentos estándar ISO: XML de OpenDocument y Open Office. También soporta los formatos propietarios como .doc y .xls. Además, posee una herramienta portapapeles web que permite a los usuarios copiar y pegar contenido entre otras plantillas de Google: documentos, hojas de cálculo, presentaciones y dibujos. El portapapeles web también se puede utilizar para copiar y pegar contenido entre diferentes ordenadores.

Los elementos copiados se almacenan en los servidores de Google durante un máximo de 30 días. Por último, vale la pena mencionar que Google Docs está integrado con Gmail y Meet, permitiendo también el uso del email, chat, calendarios compartidos y videoconferencias.

Metodologías ágiles

Se suele llamar metodologías ágiles, en comparación con la ingeniería de software tradicional, al conjunto de técnicas de desarrollo dedicadas principalmente a los sistemas complejos y al desarrollo de productos con características dinámicas, no deterministas y no lineales, donde las estimaciones precisas, planes estables y predicciones son a menudo difíciles de conseguir en etapas tempranas.

Estas metodologías se basan en la experiencia práctica de expertos de cientos de proyectos, donde las aproximaciones evolutivas han demostrado ser más eficaces que los métodos tradicionales. Muchas veces se hace referencia a los métodos ágiles como métodos de adaptación continua o métodos adaptativos. Estos se centran en aplicar rápidamente los cambios de rumbo en un proyecto, modificando estimaciones, requisitos, alcance, etc., con el objetivo de garantizar un resultado final satisfactorio.

Recordemos una vez más los principios fundamentales de Agile, expuestos en el Manifiesto por el Desarrollo Ágil de Software:

- **Individuos e interacciones** sobre procesos y herramientas.
- **Software funcionando** sobre documentación extensiva.

- **Colaboración con el cliente** sobre negociación contractual.

- **Respuesta ante el cambio** sobre seguir un plan.

Se pueden ver en estos enunciados como la cultura DevOps y Agile están íntimamente relacionados. Vamos a entrar un poco más en detalle en lo que significa adoptar una metodología ágil dentro de un equipo u organización.

Equipos ágiles

Los equipos ágiles deben ser, en la gran mayoría de los casos, equipos **multidisciplinares**; es decir, existen multitud de perfiles dentro del equipo para garantizar que se pueden ejecutar todos los tipos de tareas que tienen sentido encomendar.

Recordemos que en la mayoría de los casos desearemos tener *feedback* rápido de lo que estamos haciendo, para poder responder ante cambios inesperados y entregar software lo antes posible: por eso, puede ser una buena idea complementar al equipo de desarrollo con integrantes expertos en QA, por ejemplo. O personas tanto de front-end o back-end, que puedan dar diferentes puntos de vista sobre las tareas a tratar, dentro de su área de influencia personal.

Estos miembros del equipo son llamados desarrolladores en cuanto a que se dedican a desarrollar las tareas necesarias para conseguir los objetivos.

Adicionalmente, en un equipo ágil deberían existir las siguientes figuras, heredadas de Scrum, probablemente el modelo más extendido:

- Un responsable de producto (*product owner)* cuya tarea sea la de proveer al equipo de **metas**, en forma de nuevas características a añadir o demandas de stakeholders que le pueden haber hecho directamente, en calidad de responsable del delivery.
- Un Scrum master, figura que ayuda al equipo a seguir la metodología de la manera más estricta posible, pero también apoya en aquellos procesos en los que se necesiten ajustes. Su rol es sobre todo para facilitar que el equipo cada vez funcione mejor y eliminar todo aquello que no funcione.

La experiencia nos dice que el número de integrantes de un equipo Agile debe estar entre 3 y 9 personas, siendo quizás el número óptimo 5 o 6: este número **solo incluye** desarrolladores, es decir, las personas que están implicadas directamente y no POs, Scrum masters o stakeholders.

Esto no quiere decir que no se puedan usar equipos más grandes, pero lo más probable es que se empiece a notar la falta de agilidad, reuniones más largas (un antipatrón) o demasiadas tareas en vuelo para mantener al equipo ocupado. En el momento en que el flujo óptimo de trabajo se vea afectado, es momento de partir equipos (que podrían perfectamente compartir Scrum master, e incluso tener un backlog común).

Técnicas

Dentro de Agile existen una serie de metodologías o **frameworks** ya establecidos que establecen una serie de normas y directrices sobre cómo se debe trabajar, cómo deben fluir las tareas, cómo debe ser la naturaleza de estas… El rango es muy grande, sus diferencias bastantes y no se puede decidir si una es mejor que la otra: depende totalmente del **contexto de nuestro equipo u organización**. Algunos de los frameworks más conocidos son:

- Adaptive Software Development (ASD).
- Crystal Clear.
- Feature Driven Development (FDD).
- Kanban.
- Extreme Programming (XP).
- Scrum.

Veremos, a continuación, características de las tres últimas, por ser quizás las más comunes, aunque conviene recordar que las metodologías deben poder **adaptarse** a las circunstancias de cada equipo, siempre sin perder de vista lo que aportan para evitar apartarse demasiado del método.

Scrum

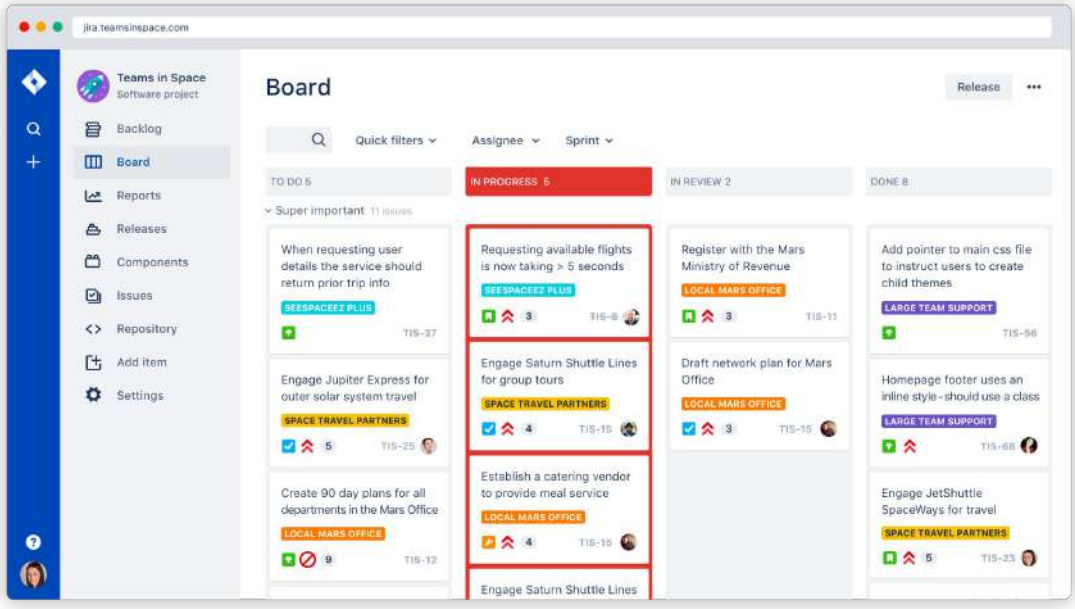

Figura 5. Fuente: https://www.atlassian.com/es/software/jira/agile

Scrum es el framework Agile más conocido y más empleado, ¡hasta tal punto que mucha gente confunde Agile con Scrum! Sabemos que esto no es cierto, nada más lejos que confundir una metodología concreta con una filosofía de trabajo que trasciende mucho más allá de la organización de tareas.

Se trata de una metodología **simple** en su concepto, que permite realizar tareas **complejas** mediante la descomposición en tareas más pequeñas y una adecuada gestión del trabajo en curso y el flujo de tareas hacia el equipo. La clave para esto reside en la **transparencia** total entre miembros del equipo para hablar de impedimentos en tareas concretas o en proponer soluciones a los problemas que hay que resolver; en la **revisión** constante de objetivos y métodos, y en la **adaptación** del equipo a los imprevistos, ajustando lo que sea necesario para conseguir el objetivo último que, como sabemos, es el de entregar valor de forma constante, rápida y sin errores.

Las tareas a ejecutar por el equipo se suelen llamar **historias de usuario** (*user stories)* y son el resultado de tomar **características** *(features)* que debe tener el producto final y descomponerlas en tareas más simples, que puedan ser acometidas por una sola persona, finalizadas en un tiempo acotado y con una clara **definición de completitud** que permita a cualquiera saber si esa tarea está lista.

Ejemplo

Una posible feature podría ser la de dotar a nuestra página de login de integración con Google, de modo que los usuarios puedan identificarse tanto con su cuenta usual, como con su cuenta de Gmail si disponen de ella.

Si tuviésemos que descomponer esto en subtareas, una posible separación sería la siguiente:

- Refactorización del sistema actual para soportar múltiples proveedores de identificación.
    - El resultado debe ser que la página sigue funcionando exactamente igual que antes, aunque si se envía cierto parámetro se intentará usar un nuevo proveedor, aunque no exista.
- Implementación de la identificación Gmail a través de Oauth 2.0.
    - El resultado será una clase/interfaz o lo que sea que reciba una serie de parámetros (usuario, contraseña o cookies) y devuelva una identificación con éxito o un error.
- Implementación de un proveedor que use la nueva identificación.
    - El resultado será que podemos hacer login con dos proveedores diferentes en vez de uno.
    - Los tests existentes deben ahora contemplar esto y se deben aumentar para cubrir las nuevas casuísticas y combinaciones.
- Adecuación gráfica de la página para informar al usuario.
    - El resultado será una nueva interfaz gráfica, validada por el equipo, compatible con el nuevo sistema de login.

Sprints

Los sprints son la clave del framework: son períodos de 2 a 4 semanas durante los cuales el equipo va a trabajar **exclusivamente** en un conjunto de historias de usuario que se hayan acordado con el product owner. Las tareas elegidas están ordenadas por el orden percibido de prioridad durante la planificación del sprint y se les habrá asignado una talla o estimación de cuánto pueden llevar, llamada **puntos de historia** *(story points).*

Esta estimación viene de un proceso de deliberación por parte de todo el equipo, está dada por la experiencia del mismo y no tiene por qué ser un valor exacto de días u horas: se usa más bien para poder hacer **mediciones** de cuánto trabajo puede sacar el equipo en cada sprint y ayudar a dimensionar la cantidad de trabajo que se puede acometer.

Las tareas suelen estar resumidas en tarjetas, físicas o virtuales, y en un estado determinado: en progreso, en espera, bloqueadas, a falta de QA, finalizado… La idea de esto es poder tener un **estado visual** del estado del sprint en cada momento para poder ajustar prioridades o reajustar al equipo durante la duración del mismo si se detecta que algo no va bien.

El **objetivo** de cada sprint es tener al final del mismo, un producto viable que se pueda entregar al cliente, ya que se ha probado convenientemente por el equipo. Dependiendo del tipo de trabajo que se haga puede que no siempre haya algo entregable, pero la mentalidad debe ser la de tener siempre algo 'que se pueda enseñar', para evitar que el trabajo se prolongue durante semanas y semanas y perder la oportunidad de **recibir feedback** que permita ajustar el rumbo…

Recordemos que se pretende escapar de los proyectos de meses en los que al ir a enseñarle al cliente el resultado, este se da cuenta de que no es exactamente lo que quería.

Un concepto importante en Scrum es que **no se añade trabajo extra durante el sprint**, a no ser que se de un suceso muy urgente que requiera de la atención inmediata del equipo: problemas en producción, errores en software entregado… Tan solo en este caso se permite añadir trabajo al sprint, asumiendo a partir de ese punto que se compromete la capacidad del equipo.

La **capacidad** del equipo en un sprint es un valor **estimado**, que se refina con el paso del tiempo y que depende de:

- La disponibilidad de los miembros del equipo durante la duración del sprint: si de un equipo de 5 personas, una persona no va a estar durante una de las dos semanas que dure el sprint, la capacidad del equipo no puede ser mayor del 90%.
- La velocidad, entendida como el histórico de puntos de historia entregados en sprints anteriores: si en un sprint con tareas por un total de 30 puntos de historia tan solo se ha conseguido entregar (pasar al estado 'Finalizado') unos 20 puntos, nada debe hacer pensar que en el siguiente se puedan entregar 30.

Gestión de backlog

Las tareas que se meten en cada sprint están en el backlog, una pila literal de historias de usuario que el product owner mantiene siempre lleno de trabajo por hacer, con el objetivo de que siempre exista algo que el equipo puede hacer para entregar valor.

Cada vez que el equipo termine un sprint, las tareas no finalizadas se vuelven al backlog, y en la siguiente planificación se tomarán de ahí las necesarias, según la velocidad del equipo y de su capacidad en ese sprint, para que se trabaje en ellas durante el periodo correspondiente.

Ceremonias

Para el óptimo funcionamiento de los sprints, lo que implica que el equipo es capaz de enfocar su esfuerzo en trabajar y aportar valor, se definen una serie de ceremonias, destinadas a mejorar de forma continua el flujo de trabajo del equipo.

Dailies

Son reuniones diarias, de duración **corta o muy corta** (más de 15 minutos se considera demasiado) en las que cada miembro del equipo le cuenta a sus compañeros:

- En qué ha estado trabajando.
- Qué problemas se ha encontrado haciéndolos.
- Qué va a hacer durante la jornada.

Si la reunión se lleva a cabo siguiendo estas premisas se convierte en una ocasión única para que todo el equipo se ponga al día del estado de tareas en las que no ha trabajado y se detecten posibles problemas que puedan surgir a futuro.

Planning

La planning es una reunión que tiene lugar al principio del sprint, en la que se decide qué tareas del backlog (o del sprint anterior) deben entrar en el siguiente y por qué. Además, es cuando todo el equipo aprovecha y hace estimaciones de puntos de historia, lo que hace que todo el mundo tenga que opinar de todas las tareas, incrementando así el conocimiento que se tiene de lo que se está haciendo y para qué.

Reviews

La review tiene lugar al final del sprint, es el momento en que se comprueba qué se ha entregado y sobre todo por qué no se ha podido entregar aquello que no esté terminado: este proceso da información valiosísima de posibles bloqueos o complejidades que no se tuvieron en cuenta durante la planificación, lo que ayudará a que siguientes sesiones de estimación sean más precisas ya que se cuenta con mucha más información.

Retrospectivas

La retro es una reunión que tiene lugar después de cerrar un sprint y su objetivo es evaluar qué cosas se han hecho bien y en cuáles se puede mejorar. Gracias a estas sesiones, llevadas a cabo con muy diferentes dinámicas, se 'hace equipo' y se sacan temas que en el día a día no suelen salir, como bloqueos, fricciones, relaciones conflictivas con equipos externos… Es una reunión clave para que se puedan entender qué cosas no funcionan y tratar de rectificarlas lo antes posible.

Kanban

Figura 6. Fuente: commons.wikimedia.org

Tal y como se explica en la web de Kanban, la historia de los tableros Kanban comienza en 1963 en el piso de manufactura de la Toyota Motor Company. Allí, Tachi Ohno trabajó en los principios básicos del método Kanban, que incluía tarjetas Kanban como señales visuales que ayudaban a controlar el flujo de piezas a través de la cadena de suministro.

Alrededor del 2006, a medida que el método Kanban iba ganando popularidad, los creadores de programas informáticos comenzaron a aplicarlo a la práctica común de visualizar y compartir el estado de los proyectos colocando tarjetas en pizarras en la sala de proyectos. La aplicación de los principios del método Kanban a los, por demás caóticos, tableros de proyectos, les dio las columnas y la estructura de las que han surgido los tableros Kanban modernos. Ahora, se utilizan con éxito en todo tipo de industrias y en todos los lugares donde la visualización del estado del trabajo ayuda a organizar mejor el trabajo.

Kanban es una metodología que simplifica el trabajo diario del equipo respecto a Scrum, mediante unas premisas fundamentales:

- Se simplifica al máximo el tablero de tareas, existiendo tan solo tres posibles estados.
  - Pendiente (to do).
  - En curso (doing, in progress).
  - Terminado (done, accepted).
- Se limita el **trabajo en curso** al imponer un límite de tareas en curso de forma simultánea. No se pueden empezar más tareas hasta que no se terminen estas.
- Se elimina el concepto de sprint, se asume que el flujo entrante de tareas será suficiente para mantener al equipo ocupado.

Como vemos, cambia bastante el enfoque con Scrum, donde el sprint es el eje fundamental que guía las iteraciones del equipo y genera unos entregables periódicos que aportan valor al negocio.

XP

La programación extrema (*extreme programming*) es una metodología de desarrollo de software ágil, concebida en 1999 por Kent Beck y con el objetivo específico de promover la aplicación de prácticas de ingeniería a la creación de software. Está descrita en profundidad en el libro Extreme Programming Explained del citado autor, aquí nos centraremos en comentar las partes más importantes.

El objetivo fundamental es el de mejorar la calidad del software resultante, reducir los ciclos de feedback y disminuir los errores humanos durante el desarrollo… ¿Os suena de algo? Hemos hablado de estas metas constantemente a lo largo de todo el curso y presentado multitud de herramientas que nos ayudan a conseguirlo: XP es una más de esas herramientas, en este caso en la forma de un framework de trabajo, basado en una serie **valores, principios y prácticas.**

Antes de pasar a enumerar algunos de estos muy numerosos elementos, conviene recordar un hecho muy importante:

Scrum **no fue concebido como una metodología de desarrollo** de software.

El hecho de que se haya adaptado Scrum de forma masiva en la industria no es porque sea idóneo para ello, sino porque el método de trabajo en equipo se ajusta razonablemente bien a organizaciones en las que el software es su principal valor. Sin embargo, XP **sí es** una metodología específica de creación de software, por lo que debemos considerar aplicarla en nuestros equipos si estamos de acuerdo con la manera de trabajar que propone.

XP también es una **disciplina**, en el sentido de que o bien se asumen los principios, se comparten sus valores y se siguen las prácticas, o bien no podemos decir que estemos 'haciendo XP', con el perjuicio que eso conlleva.

Planning

XP tiene también una ceremonia de planificación, muy similar al equivalente Scrum: durante esta planificación el equipo desgrana las tareas a realizar en historias que puedan ser ejecutadas por el equipo durante la semana y comenta sobre aquellas que se han terminado para hacer una revisión de las mismas y aprender de posibles errores.

También existen las dailies, reunión **fundamental** para cualquier equipo, independientemente de si usan algún framework o no.

Principios

Los principios fundamentales en XP son:

- Feedback rápido

Como sabemos, obtener feedback lo más rápido posible de que algo no va bien es la mejor manera de poder ajustar el rumbo para conseguir el objetivo final. Existen varios tipos de feedback: de código fuente (una batería de test), de cliente (en forma de aceptación del producto o test automáticos de aceptación) e incluso del equipo, relacionado con la manera en que este se adapta a la planificación de las tareas.

- Mantener la simplicidad

Como se comentó en la lección sobre *clean code*, un código simple es un código fácil de mantener. La simplicidad no solo conlleva seguir los principios enunciados en su momento para la creación de software de calidad, sino también la adecuada elección de tareas a realizar o características a añadir al producto final, si resulta que no se necesitan aún o no aportan tanto valor como podrían ser otras.

- Aceptar los cambios

En el mundo del software los cambios son la norma, y más aún cuando se trabaja creando un producto para un cliente: los requisitos cambian o evolucionan, los planes se modifican, las prioridades se desplazan… Lo importante es asumir que esto es parte del proceso y ser capaces de aceptarlo y tomar medidas para que los cambios no impacten en nuestra forma de trabajar.

Prácticas

Dentro de XP existen una serie de prácticas o **reglas** que imponen una manera de trabajar, vamos a intentar resumirlas y agruparlas:

- Test-first

La práctica de TDD es obligatoria en XP: ninguna funcionalidad debe existir sin test que la verifiquen y documenten y como sabemos la mejor manera de conseguirlo es creando primero los test e ir añadiendo el código de producto que los hace funcionar.

- Pair programming

Para fomentar la colaboración y fomentar que el conocimiento sobre el código fuente se distribuya por todo el equipo, en XP se trabaja en parejas. Estas parejas deberían estar trabajando con **un único teclado y pantalla** e intercambiando roles frecuentemente: uno será el encargado de escribir el código, el otro apoyará el proceso creativo, aportará ideas y aportará un punto crítico a las decisiones tomadas por el primero que, en un escenario tradicional, no serán rebatidas por nadie hasta que llegue la revisión final del código. Las parejas no son fijas, deberían cambiarse cada cierto tiempo, por cada nueva feature, cada semana…

Conseguir un trabajo en pares efectivo es **difícil**, requiere mucha disciplina y puede ser percibido como una pérdida de tiempo, pero la realidad es que mejora la calidad del código resultante a largo plazo con el añadido de que dos personas del equipo conocen una característica concreta, que además ha seguido un proceso creativo mucho más completo que si hubiese sido una sola.

- Integración continua

El código debe integrarse rápido en la rama principal, con el objetivo que ya conocemos de tener siempre un producto desplegable en la rama principal de nuestro repositorio. Si hemos seguido un proceso de desarrollo con las garantías que da una base de testing sólida, lo más probable es que podamos aspirar a un proceso de CICD sólido y fiable.

Como ya sabemos, conseguir llegar a un nivel de madurez DevOps tal que nuestros proyectos tengan integración y despliegue continuos es complicado, pero las ventajas que obtenemos son tan grandes que vale completamente la pena intentarlo.

Adopción DevOps

Para terminar el curso me parece apropiado introducir un área que tiene mucho sentido en organizaciones de cierto tamaño, cuya labor principal consiste en promover no solo la cultura DevOps (que como sabemos es un esfuerzo transversal que implica a toda la organización) sino, y más importante, **procesos y herramientas** que pueden estar centralizados en algunos equipos que tienen mayor experiencia, mayor capacidad o mayor llegada a otros equipos, por ejemplo.

La **adopción DevOps** es el área que se encarga de potenciar la transformación DevOps mediante el **fomento, formación y apoyo** en el uso de las herramientas y procesos en los que esta se apoya a todos los equipos de la organización.

Herramientas

A lo largo del curso hemos conocido muchas herramientas que dan soporte a las tareas DevOps, herramientas que tienen un coste de mantenimiento no trivial cuando se trata de soluciones on-premise o cuyo hosting pertenece a la organización:

- Sistemas de integración continua

A medida que aumenta el tamaño de la organización aumentará la necesidad de recursos para construcciones, másteres para diferentes equipos… Esto supone una necesidad de gestión y manejo de infraestructura muy elevada.

- Control de versiones

Si bien la mayoría de soluciones con hosting propio (GitLab, Github Enterprise, Bitbucket…) se mantienen bastante bien, la gestión de usuarios y grupos, así como la del almacenamiento, es compleja.

- Monitorización

La suite de herramientas de monitorización de equipos y sistemas varía ampliamente entre organizaciones y en la mayoría de los casos vienen con una necesidad de integraciones con las aplicaciones que hace que para llegar a explotar sus capacidades haga falta poder llegar hasta los equipos directamente.

- Kubernetes

Como hemos dicho, la potencia de esta herramienta para gestionar los despliegues de una organización es casi proporcional a su complejidad de manejo.

Procesos

Además de las herramientas relacionadas con el ciclo DevOps, existen procesos asociados a fomentar el uso adecuado de las mismas, así como las diferentes prácticas recomendadas como *clean code*, gestión ágil de las tareas, integración continua, testing…

El trabajo de los equipos de procesos de DevOps consiste en acercarse a otros equipos, averiguar qué herramientas usan, qué metodologías, de qué manera llegan sus despliegues a producción, y en esencia **cómo trabajan**, para poder encontrar de qué maneras poder acelerar el cambio de paradigma hacia un modelo como el que conocemos. Ejemplos de esta labor pueden ser:

- Introducir o mejorar Scrum para cambiar la forma en la que se añaden nuevas características a los proyectos.
- Formación en tooling propio de la organización, como pueden ser wrappers para poder ejecutar tareas en Jenkins o desplegar recursos propios en Kubernetes.
- Organización de talleres para enseñar a incorporar las herramientas de tracing y métricas a las aplicaciones Java de la organización…

En general, se busca ser el apoyo que buscan los equipos y áreas de la empresa cuando se está en medio de un proceso complejo de transformación, incidiendo donde más impacto se pueda tener y tratando que nadie se quede atrás.

Equipos globales

En organizaciones pequeñas o startups es posible que no exista la capacidad de tener equipos apartados encargados de llevar el peso de centralizar toda la transformación o que al haber 'crecido' en la era del *cloud-first* o *native-DevOps* se asuma que la organización es suficientemente madura como para no necesitar una transformación de ningún tipo.

Sin embargo, en la mayoría de los casos esto no es así, y las organizaciones de cierto tamaño o con cierta antigüedad el riesgo de que la transformación sea particularmente lenta y difícil existe, y una manera de apoyarla es mediante **equipos globales,** equipos que centralizan la gestión de las herramientas que sostienen la cultura DevOps en la empresa y se encargan de ayudar a los diferentes equipos, o áreas enteras de la organización, a emprender cambios en su manera de trabajar para poder evolucionar hacia la **madurez DevOps**.

Niveles de madurez

El concepto de madurez DevOps se puede entender como un estado al que aspiran equipos que utilizan maneras de trabajar que consideramos que no son las 'apropiadas' para cumplir con las premisas de trabajo colaborativo, eficiente y orientado a resultados.

Esto implica, de alguna manera, poder medir en qué estado está un equipo en un momento determinado de tiempo en relación a una serie de baremos que podemos considerar 'parte de DevOps'.

Si bien esta clasificación va a depender de las particularidades de cada organización, a grandes rasgos podríamos evaluar el nivel de madurez DevOps de un equipo en las siguientes áreas:

- Gestión de código

En este área se consideraría si un equipo usa control de versiones o no, cómo lo hace, si se hacen revisiones de código de los compañeros o se siguen *branching models* que impongan un poco de orden en el caos habitual.

Equipos que tienen poca madurez en este aspecto son aquellos que necesitan más ayuda, pues están usando procesos antiguos, rígidos e incómodos que les hacen ir más lento que sus competidores, por definición.

Un alto nivel de madurez en gestión de código lo tendrían equipos que han asumido el uso de Git para trabajar de forma colaborativa, usan pull requests y code reviews de forma habitual, y han establecido modelos de trabajo como GitFlow o Trunk-based que les permiten manejar las releases de una forma mucho más cómoda.

- Construcciones

No es raro que equipos de desarrollo no tengan builds automatizadas de sus aplicaciones y no debería extrañarnos que para generar una release se tenga que hacer a mano desde el equipo de una persona concreta del equipo.

Un nivel alto de madurez en este área indica que se usan herramientas como Jenkins para generar versiones de forma automática en base a eventos en control de versiones, ejecutando todas las suites de test que se hayan definido, así como otro tipo de controles automáticos, sin intervención humana.

- Despliegues

Algo similar a lo que ocurre con las construcciones: equipos poco maduros harán los despliegues a mano o con automatizaciones poco trabajadas que no tienen margen de control. Y es probable que una marcha atrás se convierta en un drama si no se han tomado medidas.

Sin embargo, equipos más avanzados habrán incorporado los despliegues a su servidor de CICD, con los controles necesarios para saber si el servicio está funcionando de forma satisfactoria en base a métricas, que además nos ayudarán a hacer una marcha atrás automática si algo no sale como se espera.

- Testing

El testing en muchos equipos es un lujo, ya que o bien no cuentan con espacio en el día a día para crear test, o las aplicaciones son tan gigantes y acopladas que resulta casi imposible hacer ningún tipo de batería.

La madurez en testing es muy complicada de conseguir en ciertos proyectos y va muy de la mano de procesos como refactorizaciones, cambios de arquitectura y hasta de tecnología o lenguaje de programación. El objetivo debería ser el de tener confianza plena en nuestros test automáticos para saber que **cualquier cambio**, por pequeño que sea, no tendrá consecuencias inesperadas sobre el estado general de nuestra aplicación o de nuestros sistemas.

Pero llegar a ese punto no es fácil e implica tener test de todo tipo (unitarios, funcionales, end-to-end, rendimiento…) en proyectos que no habían sido pensados originalmente para ello.

- Seguridad

La seguridad es un área de importancia crítica y como tal debe considerarse dentro de las áreas de evaluación de la madurez de un equipo, y no algo a añadir *a posteriori* cuando todo el trabajo ya esté hecho.

Equipos con madurez en seguridad van a contar con los expertos en seguridad desde el inicio de cada nueva funcionalidad para no tener que hacer cambios al final del proyecto y cuentan con procesos automáticos para revisar posibles puntos flacos (como la suite de pruebas Owasp o análisis con Kali Linux | Penetration Testing and Ethical Hacking Linux Distribution).

- Planning

En el área de planning se pueden hacer las cosas de muchas maneras y hemos introducido en este tema unos marcos de trabajo Agile que muestran que se pueden reorientar los procesos de planificación para aumentar la eficiencia en las entregas de los equipos.

Un bajo nivel de madurez en este área indica que el equipo no tiene una estrategia que permita hacer entregas frecuentes, que siempre aporten valor y estén libres de errores. Más al contrario, sus tiempos de release pueden irse a semanas o meses, la cantidad de errores en las mismas es alta o muy alta, y no existe gobierno sobre la entrada de tareas al equipo.

En cambio, si un equipo utiliza metodologías ágiles en su día a día y cambia la manera en que se relaciona con los stakeholders conseguirá, como hemos visto, un flujo de trabajo mucho más optimizado, con realimentación y mejora constantes.

Roles en adopción DevOps

Lo que nos queda para terminar de definir la adopción DevOps son los roles relacionados con estas herramientas y con todos estos procesos de fomento y colaboración.

Evidentemente, los equipos encargados de la gestión y mantenimiento de las herramientas globales contarán con:

- Miembros con experiencia en operaciones, cuya labor es la de mantener el servicio en condiciones óptimas de ejecución en un entorno que exigirá una disponibilidad muy alta y un elevado ritmo de actualizaciones.

- Desarrolladores que aportarán mayor potencia para evolucionar las herramientas y facilitar las integraciones que requerirán los equipos que quieran interactuar con esas herramientas, así como facilitar las operaciones mediante nuevas evoluciones de software.

Estos equipos multidisciplinares seguirán sus propios ciclos de desarrollo, como cualquier otro equipo de la empresa, pero es obvio que tanto por su objetivo como por su organización es un equipo DevOps de manual, llenos de retos y de problemas muy interesantes por resolver.

Developer relations

Además, esto abre una puerta muy interesante: es necesaria una figura que se encargue de promocionar esas herramientas entre los diferentes equipos que actuarán como clientes de las mismas. Pensemos que las integraciones pueden ser difíciles, como en el caso de las técnicas de monitorización o despliegues en Kubernetes.

Si las integraciones son difíciles, corremos un gran riesgo de que nuestros usuarios no las usen o lo hagan mal. Y eso no sería bueno para el equipo, pues verían cómo su trabajo no es aprovechado al máximo por el entorno de la empresa, un escenario que no es bueno para nadie.

Para ello, surgen los equipos de **developer relations** *(DevRel)* cuyos miembros tienen la fundamental labor de promover esas herramientas, ser los expertos en la organización sobre ellas y actuar como **referentes** para todos esos clientes y ayudar en la adopción de esas tecnologías.

Algunas de las tareas de estos entusiastas*, advocates* o también llamados *evangelists* (término que no comparto por las inevitables connotaciones religiosas que no aportan nada al rol) son las siguientes:

1. Organización de eventos relacionados con las herramientas, para darlas a conocer, comunicar nuevas características o releases y poner en contacto a sus usuarios.
2. Workshops y talleres prácticos.
3. Colaboraciones puntuales con clientes, pasando a formar parte del equipo durante una o varias iteraciones.
4. Creación de documentación y repositorios de prueba, para su uso como referencia por los equipos clientes.
5. Comunicación constante de feedback al equipo responsable, feedback recogido de primera mano por su contacto directo con los usuarios.
6. Participación, pasiva o activa, en planificación y hasta realización puntual de tareas con el equipo global, con el fin de acercar su experiencia de primera mano.

Como veis, este rol es **fundamental** en la organización y sinergiza con los equipos globales de forma que entre ambos aportan un grandísimo valor a la transformación DevOps de la empresa.

LA UNIVERSIDAD EN INTERNET | FORMACIÓN PROFESIONAL

PROEDUCA