Tradicionalmente el proceso de actualizar una aplicación en un entorno de producción ha sido considerada una práctica de alto riesgo por el riesgo de sufrir regresiones en el sistema. Esto ha tenido el efecto negativo de alargar las entregas por meses, incrementando el *time to market* de las empresas, y reduciendo sus posibilidades frente a la competencia. Hay dos motivaciones principales para la entrega continua:

- Reducir el *time to market*: de esta manera obtenemos *feedback* de nuestros usuarios tan pronto como es posible, disponiendo de más datos para mejorar el producto. Además, los usuarios tienden a usar el producto que primero les da la funcionalidad que están buscando.

- Reducir el coste de los errores del software (*Fast Fail*): cuanto antes detectemos un error en el ciclo de desarrollo del software, más fácil será identificarlo y corregirlo.

El concepto de *Fast Fail* siempre ha traído controversia. Algunos autores discuten que es mejor mejorar el desarrollo del software para que no se produzcan fallos en producción, pero la realidad es que esto no es posible. Veámoslo con un ejemplo:

Imaginemos una empresa que lleva 3 meses preparando una release. Todo está preparado para actualizar el entorno de producción. A las 9AM se actualiza el entorno de producción, y después de comprobar que todo parece funcionar correctamente, se inicia la campaña de marketing destinada a captar más usuarios. Minutos más tarde de haber actualizado el entorno de producción, se produce un fallo en cascada que tira abajo el entorno de producción al completo. Después de diez horas de duro trabajo por parte del equipo de desarrollo y de operaciones, descubren que una función no estaba cerrando conexiones a la base de datos, y que esto había derivado en la imposibilidad de aceptar nuevas conexiones para las peticiones de usuario. Una vez solucionado el problema, la empresa hace un análisis *post mortem* para ver cómo evitar este tipo de problemas en el futuro. Las propuestas que surgen de este análisis son:

- Más pruebas manuales.
- Más pruebas automáticas.
- Revisiones de código y programación en parejas.
- Más planificación al inicio.

Ninguna de estas propuestas son una solución definitiva. Las pruebas manuales son de la largo la peor solución. Aquí aplica aquello de que “si estás haciendo más de dos veces lo mismo, algo estás haciendo mal, mejor haz un programa que resuelva el problema”. Pero tampoco las pruebas automáticas, aunque muy recomendables (por no decir indispensables), van evitar la presencia de errores en producción. Ninguna prueba automática se tan brutal, aleatoria, maliciosa, ignorante o agresiva como la suma de todos los usuarios del sistema. Las revisiones de código y la programación en parejas son también prácticas muy recomendables y reducen la cantidad de errores porque dos ojos ven más que uno, pero siempre habrá errores que sigan sin detectarse. En cuanto a una mayor planificación, puede ayudar a evitar urgencias de última hora, donde la presencia de errores aumenta considerablemente, pero tampoco

garantizan la no existencia de errores. La realidad es que este tipo de errores van a incrementarse a medida que crezca el producto y se vuelva más grande y complejo. Veamos porqué la entrega continua es la única solución escalable.

En un entorno de entrega continua, nuestro desarrollador realiza el *commit* de la función que no cierra las conexiones con la base de datos. El commit llega a producción y minutos más tarde hay una alerta indicando que producción se ha caído. Se comprueban los últimos commits y rápidamente se detecta donde está el problema, haciendo un rollback al commit anterior al que introdujo la regresión. Nuestro desarrollador prácticamente no pasa tiempo depurando ya que el error está localizado e implementa rápidamente el *fix* de su código. Este error causó una falla en cascada, pero el tiempo de caída fue mínimo.