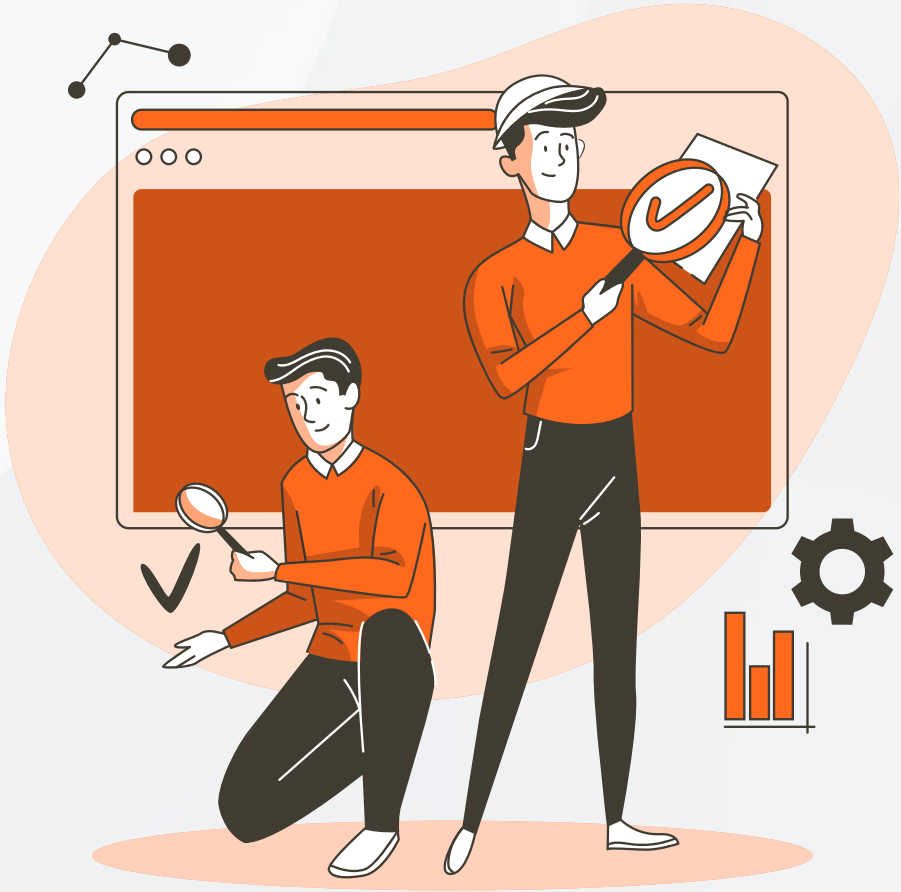
30 Consejos QA

¡Transforma tu desarrollo!



redsauce
SOFTWARE QUALITY

ÍNDICE

redsauce
SOFTWARE QUALITY

INTRODUCCIÓN

¿Te gustaría obtener mejores resultados en tus proyectos optimizando recursos y mejorando el Time-to-market?

En este ebook encontrarás **30 consejos y estrategias** prácticas que te ayudarán a enfrentar los obstáculos más comunes en el desarrollo de software, para alcanzar niveles más altos de excelencia.

Hemos creado este ebook pensando en los desafíos que te encuentras cada día. Ya seas desarrollador, responsable de producto o de calidad.

Prepárate para descubrir las claves que te harán llegar al siguiente nivel. ¡Comenzamos!

3 CONSEJOS

VALIDACIÓN Y CREACIÓN DE HISTORIAS DE USUARIO

Aprenderás cómo involucrar a los usuarios clave desde las primeras etapas del desarrollo de software, permitiéndote comprender y abordar sus necesidades de manera más precisa.

Con consejos prácticos y ejemplos, este capítulo te guiará para mejorar la calidad de tus historias de usuario, optimizando así el proceso de desarrollo y entregando un software que realmente satisfaga las demandas de tus usuarios.

1. Uso de guías o criterios de aceptación para validar historias de usuario

Los Criterios de Aceptación son un conjunto de condiciones que describen el comportamiento esperado una vez que se ha desarrollado la historia de usuario. Son los requisitos con los que evaluar si el desarrollo coincide con nuestras expectativas desde la perspectiva del usuario final.

Proporcionar un buen criterio de aceptación aumenta la confianza de que el producto refleja nuestras intenciones, reduce la posibilidad de defectos y acelera el desarrollo. Gracias a ellos los rechazos en el feedback, una vez en producción, deberían ser menos frecuentes.

CONSEJO

Debemos disponer de una herramienta para asegurar que la Historia de Usuario se ha desarrollado correctamente. Los Criterios de aceptación son una gran ayuda para este fin porque nos permite definir mejor la Historia de usuario y ofrece un criterio de validación de la misma.

Proporcionar un buen Criterio de Aceptación aumenta la confianza de que el producto refleja nuestras intenciones, reduce la posibilidad de defectos y acelera el desarrollo, ya que los rechazos en las Pruebas de Aceptación y una vez en producción son menos frecuentes.

Ningún trabajo se debe considerar realizado si no cumple con todos los Criterios de Aceptación.

2. Creación de las historias de usuario bajo los principios INVEST

INVEST es un mnemónico creado para recordar las características de un item del backlog (PBI por sus siglas en inglés) de calidad. Este item generalmente tiene formato de Historia de Usuario, cumpliendo los siguientes aspectos:

I: Independent. Debe tener nula o mínima dependencia con otras Historias de usuario.

N: Negociable. Su contenido puede ser modificado, ampliado o eliminado en función de las necesidades de negocio.

V: Valuable. Ha de aportar valor claro al producto, de manera directa o indirecta.

E: Estimate-able. Debemos ser capaces de darle un tamaño, hacer una estimación de esfuerzo requerido para llevarla a cabo.

S: Small. Ha de poder realizarse en horas, idealmente dentro de una jornada laboral. Esto ayuda a la Integración continua y a tener feedback rápido. Si el tamaño es muy grande, deberíamos poder dividirla en trozos más pequeños.

T: Testable. Ha de ser posible realizar pruebas sobre ella para validar que su comportamiento es el esperado.

CONSEJO

Seguir los principios INVEST (Independent, Negociable, Valuable, Estimate-able, Small, Testable) a la hora de crear las Historias de Usuario nos permite dotarlas de características que nos ayudarán en su desarrollo. En caso de no seguirlos, podemos encontrarnos con:

- Dependencia entre historias de usuario, retrasando entregas, cambiando alcances, etc.
- Contenido no negociable, haciendo el desarrollo muy rígido y difícil.
- Resultados de poco o nulo valor, siendo difícil el responder la pregunta, ¿por qué es necesaria esta HU?
- HU poco definidas, complejas de estimar, que deberán definirse a posteriori con la carga de trabajo extra que implica.
- Tareas extremadamente grandes, que bloquean a miembros del

equipo durante días, ralentizan las revisiones de código, dificultan la integración continua, etc.

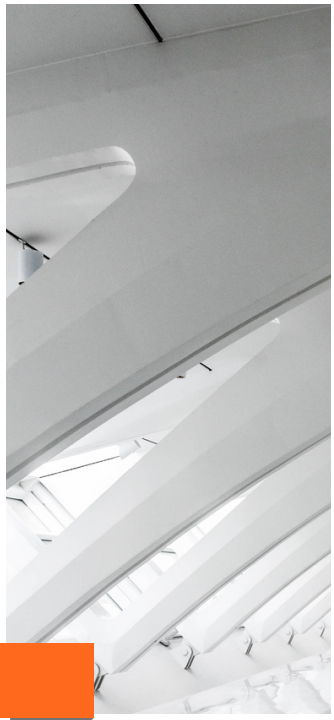
- Dificultad o imposibilidad de probar la HU, lo que aumenta la probabilidad de que aparezca un error tras el despliegue o en un futuro (detectable por pruebas automáticas).

3. Establecimiento de la DoR (Definition of Ready) para crear las historias.

El Definition of Ready o en castellano, Definición de preparado o Definición de listo, es un acuerdo entre el equipo y el Product Owner sobre lo que significa que un item del backlog (usualmente llamado PBI) esté listo para ser entendido, valorado y ejecutado.

CONSEJO

Disponer de un Definition of Ready agiliza el desarrollo de los items del backlog al ayudar a tenerlos definidos antes de comenzar su programación.



4 CONSEJOS

HERRAMIENTAS PARA GARANTIZAR LA CALIDAD DEL CÓDIGO

Exploraremos diversas herramientas que te ayudarán a mejorar la calidad del código en tus proyectos de software. Hablaremos de herramientas de análisis estático, pruebas unitarias y TDD para identificar y corregir errores en tu código.

Estas herramientas te permitirán mejorar la legibilidad, mantenibilidad y eficiencia de tu código, asegurando un producto final de mejor calidad.

4. Herramientas de análisis estático de código

El análisis estático de código permite evaluar el código desarrollado sin necesidad de ejecutarlo. Es una prueba que debería realizarse automáticamente antes de que el código desarrollado en una rama se acepte en otra. Habitualmente es el servidor de integración continua quien tiene un trigger que lo lanza. Se trata de una prueba rápida de ejecutar y que aporta un gran valor a la integración continua. Proporciona un gran ahorro en coste al anticipar problemas y errores no funcionales.

CONSEJO

Recomendamos disponer de una herramienta para validar el código estático de una rama previamente a una integración. La configuración no es compleja y la obtención de información relevante es muy rápida. Con esta información además se pueden proponer diferentes políticas de “quality gates” y dotar a las pipeline de desarrollo de criterio para incluso detenerlas si fuese necesario.

5. Uso de Linters en el IDE del desarrollador

Un linter es una herramienta que analiza nuestro código en tiempo real a medida que se va escribiendo. Mediante una serie de reglas definidas por el desarrollador, ayuda a seguir las buenas prácticas o guías de estilo de nuestro lenguaje y detectar errores o warnings que podrían provocar problemas de compilación o bugs.

CONSEJO

Generar software de calidad tiene su dificultad entre otras razones por las diferentes variables que le afectan y la casi infinita manera de poder hacer una misma cosa. Recomendamos que el equipo de desarrollo emplee herramientas que permitan generar código siguiendo un estilo común,

las buenas prácticas dentro del lenguaje y en definitiva un producto con la mayor calidad posible. Configurar un linter en el IDE del entorno de trabajo es una de las mejores alternativas, es sencillo y con beneficios desde el primer momento.

6. Empleo de Test unitarios durante la integración continua

Los tests unitarios son pruebas de pequeñas funcionalidades que se ejecutan muy rápidamente, siempre de manera automática. Generalmente los programa el desarrollador con el fin de asegurarse una correcta implementación y una estabilidad futura de su trabajo. Este tipo de test son muy eficientes si están bien desarrollados, al poder ejecutarse en pocos segundos (idealmente en menos de 1 segundo) y de manera global, ofreciendo un feedback rápido, algo muy deseado en desarrollos ágiles.

CONSEJO

Posiblemente una de las mejores apuestas por la calidad en un equipo de desarrollo es el compromiso de crear, mantener y ampliar una cobertura de test unitarios. Es la base de la clásica Pirámide del Test porque nos dan feedback muy rápido y son baratos de crear y mantener, sobre todo comparándolos con los tests de User Interface.

Recomendamos encarecidamente integrar los test unitarios en el proceso de desarrollo, incluyéndolos en la estrategia de calidad.

7. Uso de metodología TDD

TDD es una práctica de programación de software en la que antes de programar la funcionalidad que queremos implementar, se escribe una

prueba (en la práctica un test unitario) que la validará. Al no haber código todavía, la prueba fallará. A continuación se desarrolla el mínimo código que hace que la prueba pase satisfactoriamente. Finalmente, ese código se refactoriza y estaría listo para darlo por terminado.

CONSEJO

Implementar TDD conlleva varias ventajas. Por un lado, el mero hecho de pensar en la funcionalidad para implementar el test y que esa sea mínima pone foco en cómo se usará, desvela casos límite y organiza el código. También reduce el esfuerzo de desarrollo en etapas maduras de la aplicación, produciendo con mayor calidad y en menor tiempo.

Si se toma la decisión de comenzar a aplicar TDD, recomendamos que se ponga en contacto con alguien que tenga experiencia para guiar en los primeros pasos.

2 CONSEJOS

GESTIÓN DEL PROCESO DE INTEGRACIÓN DE CÓDIGO EN EL REPOSITORIO

La integración de código entre diferentes ramas es un aspecto básico para asegurar la estabilidad y la calidad del software. Presentamos dos estrategias fundamentales para manejar eficientemente las ramas, resolver conflictos y garantizar una integración sin problemas.

8. Empleo de Merge request y revisión de código

Una Merge request (también llamada Pull request en algunas herramientas) es una manera de revisar, probar y aprobar un cambio de código en el repositorio. Es posible asignar la revisión a uno o varios componentes del equipo, requerir o no una aprobación bajo determinadas reglas, etc.

CONSEJO

Disponer de una política de Merge Request permitirá que el código desarrollado por un miembro del equipo pueda ser revisado por otros y reducir la probabilidad de tener errores de diseño, defectos, etc.

Otra gran ventaja de implementar Merge Request es que se trata de una gran fuente de conocimiento y formación, al compartir diferentes formas de afrontar un problema y programar la solución.

9. Disfrutar de un criterio unificado de cómo crear una Merge Request

Además de tener establecido el mecanismo de revisión de Merge Request, es posible tener un acuerdo en el equipo de desarrollo sobre las características de la Merge Request, como por ejemplo:

- Formato del título, forma de redacción, longitud máxima, uso de prefijos...
- Contenido de la descripción, formato, tamaño...
- Tamaño del contenido del código en la merge request, cantidad de cambios, número de líneas, cantidad de archivos...

CONSEJO

No disponer de unos criterios acordados por todo el equipo técnico con respecto a cómo han de ser las Merge Request puede conducirnos a:

- Dificultad para encontrar los defectos debido a que hay demasiado código a revisar. Esa Merge Request sería susceptible de ser dividida en otras más pequeñas. Esto puede llevar a no encontrar algunos defectos, al quedar “escondidos” entre tantas líneas o a complicar la validación si el cambio afecta a diferentes funcionalidades.

- Retrasar la identificación de un problema en el momento de la revisión o a la hora de hacer un análisis postmortem posterior debido a la existencia de diferentes formatos de títulos y ser poco autoexplicativos.

- Aumentar el tiempo de revisión al tener que entender qué hace el cambio o tener que preguntar, si no hay una buena descripción.

2 CONSEJOS

¿CÓMO VERIFICAR QUE SE CUMPLEN LOS REQUISITOS DE CALIDAD

Te proporcionaremos 2 estrategias para garantizar que tu producto cumpla con los estándares de calidad establecidos. Aprende a definir criterios de calidad claros y medibles, y cómo utilizar herramientas y métodos efectivos para evaluar el cumplimiento de estos requisitos.

10. Los programadores verifican los "Criterios de Aceptación"

Los criterios de aceptación (CA) son un conjunto de condiciones que describen el comportamiento esperado una vez que se ha desarrollado la historia de usuario. Son los requisitos con los que evaluar si el desarrollo coincide con nuestras expectativas.
La validación de que los criterios de aceptación se están cumpliendo puede llevarse a cabo únicamente en la propia prueba de aceptación por el Product Owner, Stakeholder o la figura que corresponda en su equipo (acercamiento "clásico") o puede llevarse a cabo en otros puntos del ciclo de vida del producto, como durante la programación de la funcionalidad.

CONSEJO

Consideramos esencial que los desarrolladores verifiquen el cumplimiento de los Criterios de Aceptación en el trabajo que están realizando. De esta manera, el feedback en caso de que algo no funcione según lo esperado lo tendremos mucho antes y la Prueba de Aceptación tiene muchas más probabilidades de resultar satisfactoria.

Como resultado, ningún desarrollo se debe considerar terminado si, entre otras cosas, no cumple con todos los Criterios de Aceptación.

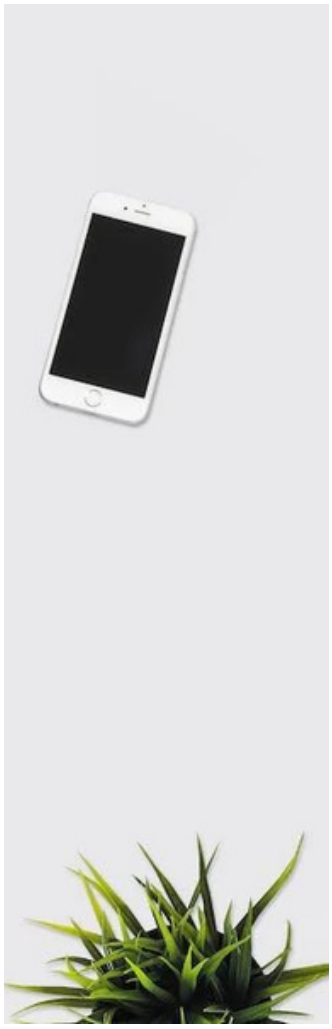
11. Establecimiento del Definition of Done (DoD)

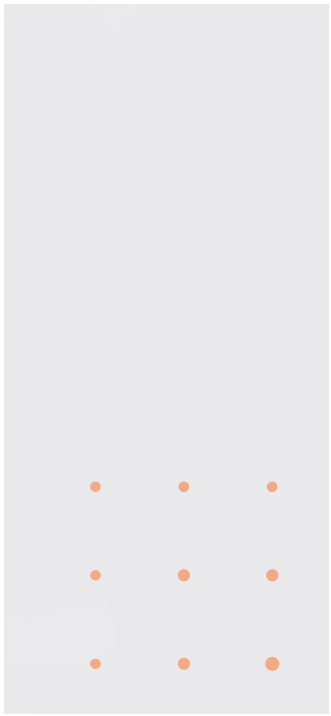
El Definition of Done o según sus iniciales, DoD, es un acuerdo al que llega un equipo de desarrollo sobre qué significa que una release esté lista para ser desplegada en Producción.

Generalmente este acuerdo engloba aspectos sobre tres componentes: requisitos funcionales (historias de usuario, criterios de aceptación...) o de negocio, calidad (cobertura de test, defectos...), y requisitos no funcionales (rendimiento, seguridad, usabilidad...).

CONSEJO

Recomendamos redactar y disponer del documento Definition of Done e ir iterando sobre él. Nos ayudará a reducir la posibilidad de que despleguemos un producto incompleto, inseguro, ineficiente, defectuoso, etc.





6 CONSEJOS

AUTOMATIZACIÓN DE LAS PRUEBAS

Descubre cómo reducir tiempo y recursos, realizando pruebas exhaustivas con frameworks y herramientas de automatización.

6 consejos prácticos que te proporcionarán las calves que necesitas. Consigue identificar y solucionar errores de manera rápida y eficiente, entregando un producto final de mayor calidad a tus usuarios.

12. Tests automáticos de integración

El test de integración es la fase en la que se prueba la interfaz de comunicación entre módulos de software que hemos podido probar anteriormente de manera aislada. De este modo, los módulos se “integran” y se prueba el conjunto como una unidad.

CONSEJO

Los test de integración son parte inexcusable de una estrategia de testing, porque reducen el riesgo de que aparezcan errores al conectar módulos que de manera aislada, hemos validado que funcionan correctamente. Recomendamos incluirlos dentro de los planes de prueba porque pese a ser más lentos de ejecutar que los test unitarios o los de contrato, nos permitirán un feedback más rápido que los test de User Interface.

13. Test automatizados de API

Las pruebas de API, o pruebas de servicio validan su funcionalidad, estructura, fiabilidad, rendimiento, seguridad, etc, formando parte de los test de integración. A la hora de automatizarlas, son pruebas más rápidas de ejecutar que las de User Interface.

CONSEJO

La automatización de las pruebas de API las englobamos en el conjunto de los test de integración de la aplicación. Consideramos muy importante tener cobertura de estas pruebas para añadir una capa de confianza sobre todo a medida que el número de APIs crece. Gracias a su rapidez de ejecución frente a las pruebas de User Interface, dispondremos de un rápido feedback del estado de la aplicación, de la posibilidad de tener una

gran cobertura funcional y de una gran facilidad para incluirlas dentro de la integración continua.

14. Test automáticos de User Interface

Hay aplicaciones que por su naturaleza, no requieren test de User Interface sencillamente porque no tienen interfaz. Sin embargo, lo más habitual es que haya una interfaz con la que interaccionamos. La funcionalidad que ofrece esta interfaz puede validarse con test manuales y con test automáticos que simulen la acción de un usuario. Debido a su naturaleza, estos test automáticos suelen ser lentos de ejecutar y costosos de mantener, por lo que es imprescindible tener un buen criterio de selección de qué funcionalidades se verán cubiertas por ellos.

CONSEJO

Si la aplicación dispone de un interfaz de usuario recomendamos realizar una batería de pruebas que validen que esa interfaz cumpla los criterios de aceptación para los que fue diseñada. Es posible que se hayan automatizado los test unitarios, de integración y de sistema que hay “por debajo” pero es posible que estos test indiquen que todo está bien y sin embargo un problema en el frontend o en la comunicación entre éste y el backend haga que la aplicación falle.

15. Pruebas de rendimiento

Los test de rendimiento son una tipología de test no funcional que evalúan el tiempo de respuesta, estabilidad, confiabilidad, escalabilidad, etc. de un sistema o aplicación bajo determinadas cargas. En estas pruebas no se validan aspectos funcionales pero permiten conocer bajo qué circunstancias el sistema va a ofrecer una mejor o peor experiencia de

uso, permitiendo depurarlo encontrando cuellos de botella, ineficiencias en el código, etc... y así poder dimensionarlo correctamente. Por su propia naturaleza, necesitan de una herramienta para poder ejecutarse.

CONSEJO

Consideramos vital para la experiencia del usuario que sistemas que vayan a estar sometidos a cargas variables de acceso se sometan a pruebas de rendimiento con el objetivo de detectar problemas no funcionales imposibles de encontrar con otra tipología de pruebas.

Por esta razón recomendamos desarrollar dentro de la estrategia de pruebas de la empresa, baterías de test orientadas al rendimiento de la aplicación y que éstas se incluyan dentro del pipeline de desarrollo.

16. Tests de regresión

Los test de regresión son un conjunto de pruebas generalmente automáticas que aseguran que la funcionalidad existente de la aplicación no se ha roto.

Abarcan toda la aplicación para asegurar que los cambios realizados en un área funcional no han afectado a otras que, sobre el papel, no se han modificado.

En general se trata de una selección de tests ya existentes y que pasan por prácticamente todo el sistema. También suelen incluir test que validan que determinados bugs encontrados en el pasado no se vuelven a reproducir.

CONSEJO

Es necesario en cada release asegurar que funcionalidades core para el negocio y que no se han modificado por el desarrollo de esa release siguen funcionando correctamente y que los defectos detectados y los históricos no se reproducen.

Recomendamos tener una batería de este tipo de test en el pipeline de desarrollo para aumentar la confianza en el producto a desplegar siendo capaces de bloquear el pipeline si se encuentra un defecto en el código.

17. Tests de rendimiento

Por su propia naturaleza, los test de rendimiento necesitan una herramienta para poder ejecutarse. Esta herramienta puede lanzarlos bajo demanda, periódicamente y/o dentro de un pipeline del servidor CI/CD a lo largo del ciclo de desarrollo.

CONSEJO

La experiencia nos ha indicado que si unas pruebas automáticas no están vinculadas a un proceso que las lance de manera periódica o bajo un trigger determinado (un commit, un despliegue...) dejan de servir para el propósito que se crearon y posiblemente dejen de usarse y queden obsoletas.

Nuestra recomendación pasa por que todas las pruebas, incluidas las de rendimiento, se incluyan en un punto del pipeline del ciclo de desarrollo para poder tener su feedback en al menos un punto del mismo.

3 CONSEJOS

ENTORNOS DE TEST

Descubre la importancia de los entornos de prueba en el desarrollo de software y cómo pueden ayudarte a garantizar la calidad de tus aplicaciones.

Hablaremos de entornos de test efectivos que imiten de cerca el entorno de producción, permitiéndote probar tu software en condiciones realistas antes de su lanzamiento.

18. Empleo de entornos de Test diferentes al entorno Productivo

Un “entorno de test” simula el entorno productivo de una aplicación. Allí podemos desplegar versiones de la aplicación con el fin de poder realizar las pruebas que la validen.

En función del uso que se le de y del tipo de pruebas a realizar, la madurez del código que aloje, etc... pueden clasificarse en categorías. Por ejemplo: el entorno de desarrollo, el entorno de testing, entorno de staging o el entorno de UAT (también llamado “preproducción”).

Con el uso de técnicas como “feature flag”, “canary releasing”, etc, los equipos de desarrollo pueden llevar a cabo ciertas pruebas directamente en Producción.

CONSEJO

El testing es esencial para cualquier metodología de desarrollo de software. Una estrategia débil de pruebas puede conllevar despliegues con errores y defectos. Para mitigarlos es necesario realizar pruebas y éstas han de ejecutarse en un entorno de pruebas dedicado.

Recomendamos tener los suficientes entornos de pruebas como para cubrir los tipos de test que requiera la aplicación que estamos desarrollando.

19. El entorno de test y la aplicación que corre en él presenta las mismas funcionalidades que en Producción

Los entornos de test se conciben para, una vez desplegada la aplicación, obtener un comportamiento y funcionalidad lo más fiel posible al de producción. A menudo, por limitaciones técnicas, los entornos de pruebas no disponen de ciertas características que sí tienen los de producción.

CONSEJO

Los despliegues en estos entornos deben seguir los mismos pasos que necesitamos para hacerlo en producción. De este modo probaremos implícitamente también los pasos de despliegue.

Es posible que haya entornos en los que falte alguna funcionalidad que si podremos encontrar en otros. Estas situaciones no deseadas debemos tenerlas controladas y hemos de establecer un mecanismo que nos permita estar seguros de que en todo momento podremos lanzar en un entorno no productivo cualquier ciclo funcional que se pueda dar en producción.

20. Activación y desactivación de entornos de Test con distintas versiones de la aplicación de manera automática

Los entornos de test se emplean para varias tipologías de pruebas. Puede ser que interese probar el último desarrollo con una base de datos similar a la productiva, que se desee realizar una UAT con el Stakeholder o que necesitemos hacer pruebas de rendimiento.

En todos los casos se recomienda disponer de una herramienta que nos permita crear y configurar un entorno desde cero, instalar una determinada release de la aplicación, tenerla operativa y accesible para hacer las pruebas, generar un informe de las pruebas y finalmente, eliminar el entorno liberando así los recursos.

CONSEJO

Las metodologías de desarrollo actuales, espoleadas por Agile, fomentan el desarrollo en grupos heterogéneos, squads, equipos Agile... Estos equipos generan versiones de la aplicación que deben ser rápidamente probadas tanto manual como automáticamente para tener un feedback ágil.

Disponer de múltiples entornos permanentes implica un alto coste fijo de infraestructura, mantenimiento, gestión etc. Por esa razón toma una gran importancia tener automatizada la creación, uso y eliminación de esos entornos de test, generalmente orquestada por el servidor de CI/CD.

1 CONSEJO

INTEGRACIÓN DE LAS PRUEBAS EN EL PIPELINE DE DESARROLLO

Debemos incorporar las pruebas automáticas en cada etapa del ciclo de vida de desarrollo, desde la construcción hasta la implementación, para asegurar la calidad del software en todo momento.

Es el sistema de CI/CD, los diferentes pipelines de desarrollo programados son los que orquestan la ejecución de estas pruebas automáticas en el momento adecuado.

21. Integración en los pipelines de desarrollo

CI/CD son los acrónimos de **Integración Continua y Despliegue o Entrega Continua**. Se trata de una práctica de desarrollo de software en la que los cambios de código en el repositorio se llevan a cabo de manera frecuente y segura (CI) automatizando los builds y las pruebas de los mismos.

De este código se pueden generar de manera automática releases listas para instalar en entornos preproductivos (C. Delivery). Muchos equipos de desarrollo se quedan en este punto, ejecutando el despliegue en producción de manera manual.

Finalmente, podemos llegar al estado de madurez en el que el código se despliega automáticamente en producción tras una serie de pasos (C. Deployment).

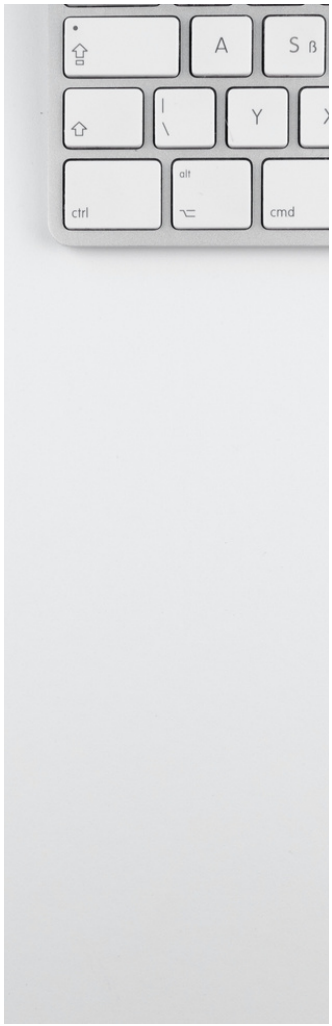
Ambos procesos de CI y CD se llevan a cabo mediante una serie de acciones sincronizadas en uno o varios pipelines, donde se lleva a cabo la descarga del código, las pruebas, despliegues, etc.

CONSEJO

Automatizar la integración del código y los despliegues a entornos pre y productivos agiliza tremendamente la entrega de valor de los equipos de desarrollo.

Adicionalmente, las pipelines de CI/CD sirven de columna vertebral para lanzar pruebas tanto funcionales como no funcionales sobre un código y validarlo como candidato a ser release para producción. De este modo las pruebas se lanzan en el lugar y momento preciso en función de su naturaleza. Las de código estático antes del despliegue, las de servicio o las de integración antes de las de User Interface... Y en el caso de que una de ellas falle, todo el proceso se puede hacer el "harakiri" y enviar las notificaciones pertinentes.

La rapidez para disponer de un entorno de pruebas o de demostración, para responder en caso de un rollback, o el empleo de los pipeline de Continuous Delivery para alcanzar Continuous Deployment son otras de las razones para recomendar el desarrollo de los pipelines de CI/CD.



2 CONSEJOS

PROCESOS DE AUDITORÍAS DE SEGURIDAD Y LA POLÍTICA ASOCIADA

Una auditoría de seguridad consta de una serie de análisis y pruebas que permiten evaluar el estado de seguridad que presenta un sistema de información frente a un ataque de seguridad.

El objetivo es detectar vulnerabilidades que permitan que un tercero pueda causar daños a la empresa.

Debido a que el código de las aplicaciones, la infraestructura de la empresa, el personal empleado etc., cambia en el tiempo, las auditorías de seguridad se han de llevar a cabo periódicamente.

22. Auditorías de seguridad periódicas

Una auditoría de seguridad consta de una serie de análisis y pruebas que permiten evaluar el estado de seguridad que presenta un sistema de información frente a un ataque de seguridad. El objetivo es detectar vulnerabilidades que permitan que un tercero pueda causar daños a la empresa.

CONSEJO

Debido a que el código de las aplicaciones, la infraestructura de la empresa, el personal empleado etc, cambia en el tiempo, recomendamos realizar auditorías de seguridad de manera periódica para asegurar que no hay una brecha en el sistema y que las acciones propuestas de medidas preventivas, refuerzos, correcciones... surten efecto.

El alcance de estas pruebas, su caracterización y periodicidad dependerán del estado actual de la aplicación y debería ser uno de los apartados de la estrategia de calidad a implementar.

23. Documentación de la política de seguridad

La “política de seguridad” según la RFC 2196 se define como: “declaraciones formales de las reglas que debemos cumplir las personas que tenemos acceso a los activos de tecnología e información de una organización”.

De este modo, afecta tanto a empleados como a proveedores o clientes. Se materializa en un conjunto de documentos que explican lo que se espera de todos ellos, para prevenir incidentes de seguridad, reduciendo la vulnerabilidad.

CONSEJO

A día de hoy es prácticamente imposible disponer de un sistema informático totalmente seguro. Sin embargo, recomendamos disponer de políticas de seguridad sencillas y claras al alcance de las personas relacionadas con la actividad de la empresa. De este modo tendremos herramientas para protegernos de accesos no autorizados, prevenir errores o descuidos de seguridad y en general, minimizar la vulnerabilidad.

1 CONSEJO

LISTADO DE NAVEGADORES / SO, SOPORTADOS POR SU APLICACIÓN

Conocer los navegadores, sistemas operativos (y sus respectivas versiones) que tu aplicación puede soportar es crucial para asegurar una experiencia de usuario óptima y maximizar la accesibilidad de tu software.

24. Listado de navegadores soportados

Es posible que el producto requiera un navegador y/o un sistema operativo para que el usuario pueda ejecutarlo.

En ese caso, el diseño del producto y las pruebas realizadas validan que ese producto cumple con los requisitos bajo unas determinadas combinaciones de navegadores y/o sistemas operativos.

CONSEJO

En caso de que el producto desarrollado requiera un navegador y/o un sistema operativo para que el usuario pueda ejecutarlo, consideramos muy interesante indicar qué combinación de navegadores y sistemas operativos están soportados.

De este modo queda claramente delimitada la línea entre lo que es un defecto que requiere una corrección de lo que es una operativa no soportada.

4 METODOS

EMPLEO DE PROCESOS DE DESPLIEGUE CONTROLADO EN ENTORNOS PRODUCTIVOS

Tener un proceso de despliegue controlado unido a una monitorización efectiva permitirá identificar de manera temprana posibles incidencias.

De esta forma, en función de su gravedad, podremos tomar decisiones oportunas: continuar con el despliegue aun con la incidencia, detenerlo, solucionarlo y volver a desplegar, realizar un hotfix tras el despliegue, etc.

25. Feature toggles

Permite desplegar en producción código con funcionalidades no terminadas o que se quieren activar bajo demanda. Esto hace más fácil que las releases suban a producción y se puedan activar funcionalidades cuando se desee para todos los usuarios o para subconjuntos de ellos.

La activación de esas funcionalidades puede llevarse a cabo sin requerir un cambio de código en producción, empleando un backoffice, una query en la base de datos, etc.

26. Canary Releasing

En el caso del Canary Releasing, el balanceador es más complejo y hace apuntar a un determinado grupo de usuarios dentro del server con la nueva release, como empleados de la empresa, betatesters, usuarios de una determinada zona, etc.

27. Blue-Green deploy

Los procesos de despliegue controlado permiten despliegues sin "downtime" (ZDD, Zero Downtime Deployment) y sobre todo aumentan la resiliencia del sistema.

El despliegue Blue-Green es el estándar y más sencillo, donde (simplificando mucho) el server "Blue" corresponde al código actual en producción y el "Green" el que contiene la nueva release. Una vez lista la release, un balanceador desconecta progresivamente la carga del Blue y la dirige al Green.

De este modo el código nuevo está habilitado para todos los usuarios de manera transparente para ellos

28. Dark Launch

Finalmente, el Dark Launch es todavía más complejo, y en él dispondremos de dos tipos de entornos productivos:

“Producción Estable” y “Producción Testing”

De este modo, en “Producción Testing” podemos activar funcionalidades únicamente para algunos usuarios o incluso que estén allí pero no sean visibles, como un botón oculto que se active al cargar la página y que nos permita evaluar el rendimiento de su transacción de manera totalmente transparente al usuario.

1 CONSEJO

DEFINIR MÉTRICAS DE CONTROL DE CALIDAD EN LA APLICACIÓN

Desde hace más de 100 años se atribuye a Lord Kelvin la frase:

"Lo que no se define no se puede medir. Lo que no se mide no se puede mejorar. Lo que no mejora, se degrada siempre".

Establezcamos una base sólida de control de calidad en tu aplicación mediante la definición de métricas claras y relevantes.

29. Métricas de control de calidad

En el ámbito de la calidad de software es necesario tomar decisiones para mejorar algo que se está degradando o está en riesgo de ello.

Estas decisiones se deben realizar en base a unos sistemas de información basados en indicadores que nos permitan planificar, establecer objetivos, controlar los resultados, etc.

Algunos ejemplos pueden ser el DLR (Defect Leaking Ratio), MTTR (Mean Time To Recovery), MTTA (Mean Time to Acknowledge), etc

CONSEJO

La mejora de la calidad es un proceso vivo, que se retroalimenta de datos para mantener una mejora continua. Si no disponemos de los indicadores adecuados no tendremos información para articular palancas de mejora.

Por esta razón recomendamos realizar un estudio de los puntos más débiles del ciclo de desarrollo y determinar métricas SMART con las que establecer unos KPI en base a los cuales podamos trabajar y evaluar resultados.

1 CONSEJO

HABILITAR PROCESOS DE MONITORIZACIÓN, EVALUACIÓN Y PRIORIZACIÓN DE INCIDENCIAS EN PRODUCCIÓN

Evaluar y categorizar las incidencias de acuerdo a su impacto y urgencia, permite priorizar y asignar recursos adecuadamente para su resolución.

Esto garantiza una respuesta rápida y eficiente a los problemas que surjan en producción y minimiza su impacto en los usuarios

30. Monitorización en tiempo real con un criterio común de evaluación de prioridad

Un sistema de monitorización realiza un seguimiento de las actividades que realizan usuarios, aplicaciones y otros servicios de nuestra aplicación. De este modo, podemos supervisar todos los procesos que se llevan a cabo y mostrar los resultados en diferentes paneles de mando, informes, etc.
En el caso de que se degrade la experiencia del usuario por defectos en el código, fallos de hardware o de red, agotamiento de recursos, errores de configuración, inconsistencia de datos, etc, la monitorización dispone de un sistema de alertas que activan protocolos para solucionar la incidencia.

Las incidencias en el entorno Productivo pueden llegar por diversas fuentes: un sistema de monitorización, detección por parte del equipo interno, por feedback del equipo de soporte a los clientes, etc.
En todos los casos, será necesario reflejar la incidencia como una tarea a resolver y caracterizarla. Esta caracterización generalmente pasa por asignar una “Prioridad” y, opcionalmente, una “Criticidad”.

El equipo técnico generalmente tiene visión para asignar la criticidad de una incidencia pero es el equipo de Producto quien habitualmente tiene el criterio para asignar la prioridad (los casos extremos son una excepción), apoyado en el equipo técnico.

Este criterio puede estar definido, compartido y consensuado o puede no estarlo.

CONSEJO

Recomendamos que, dentro del sistema de monitorización, se configuren diferentes alertas en función de la naturaleza de las incidencias.
Esto permitirá entre otras cosas organizarlas por prioridad o notificar a diferentes equipos en base a tu tipología.

En caso de partir de cero, la sugerencia es comenzar por aquellas incidencias que puedan afectar a la seguridad, a la imagen de marca o experiencia de usuario, las que puedan implicar un perjuicio económico o las que faciliten la actividad al equipo de desarrollo.

Estas alertas, una vez confirmada la incidencia deben priorizarse, convertirse en tareas e incorporarlas al proceso de desarrollo.

Una de las razones de posible tensión entre el equipo técnico y el de producto es el criterio de priorización de las incidencias.

Para reducir esa tensión nuestra recomendación es definir, compartir y consensuar un procedimiento en el que se reflejen de manera lo más objetiva posible los diferentes criterios y valoraciones que se realizan a una incidencia para asignarle una prioridad.

Habrá ocasiones en las que esto no será necesario pero otras en las que ese documento ayudará a eliminar ambigüedades de criterio.

CONÓCENOS

SOBRE NOSOTROS

Somos **especialistas en automatización de pruebas** funcionales y su integración dentro del ciclo de desarrollo de software desde 2004.

Ayudamos a nuestros clientes, reduciendo su time to market y aumentando la confianza en sus despliegues gracias a los procesos de SQA.

- Reducimos el tiempo dedicado a las pruebas del producto, disminuyendo costes.
- Identificamos tus cuellos de botella, antes de salir a producción.
- Actuamos en todo el SDLC para mejorar la calidad de forma global.
- Analizamos tu producto para detectar ineficiencias o vulnerabilidades.

¿TRABAJAMOS JUNTOS?

Cuéntanos tu caso y te responderemos a la mayor brevedad posible.
Crearemos un plan de acción y llevaremos tu desarrollo al siguiente nivel.
Envíanos un mail a:

info@redsauce.net

www.redsauce.net

Pablo Gómez

Cofundador de Redsauce