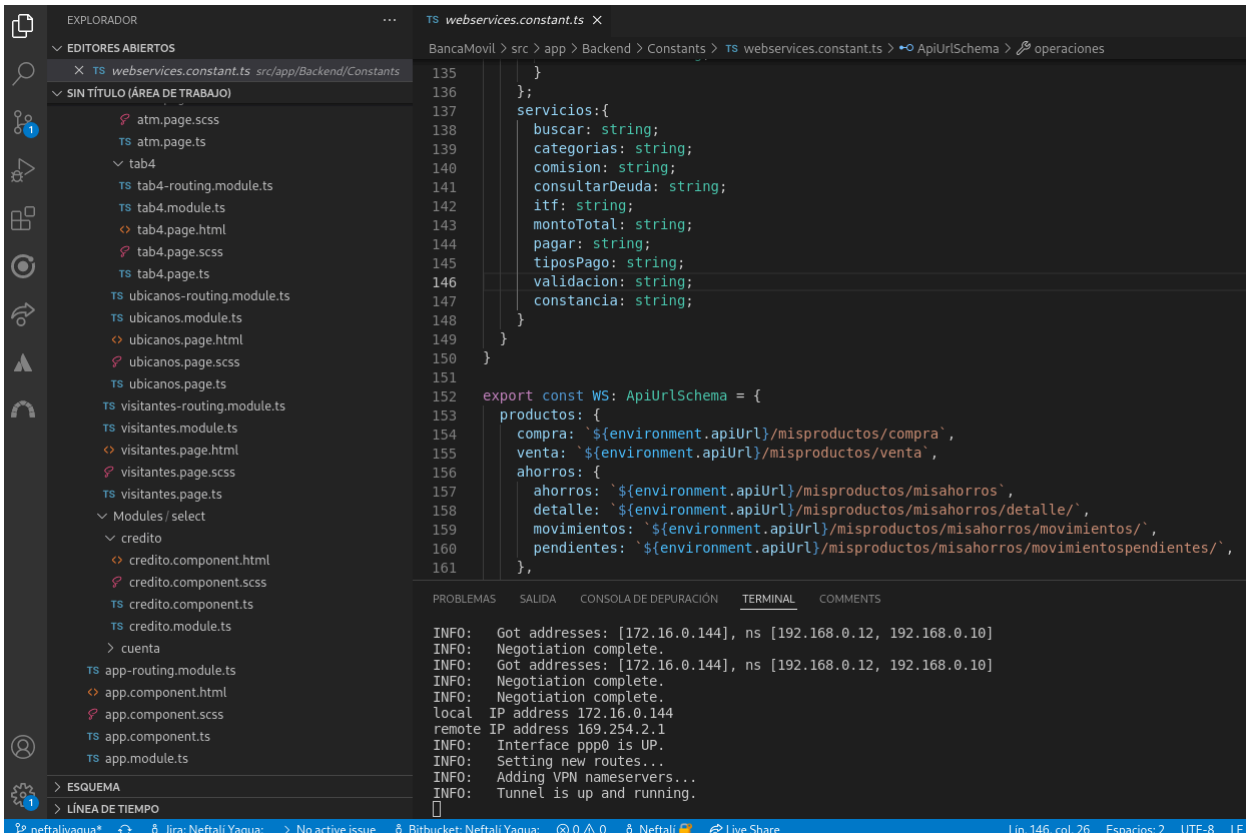



# Entornos, versiones y ciclos de lanzamiento del software

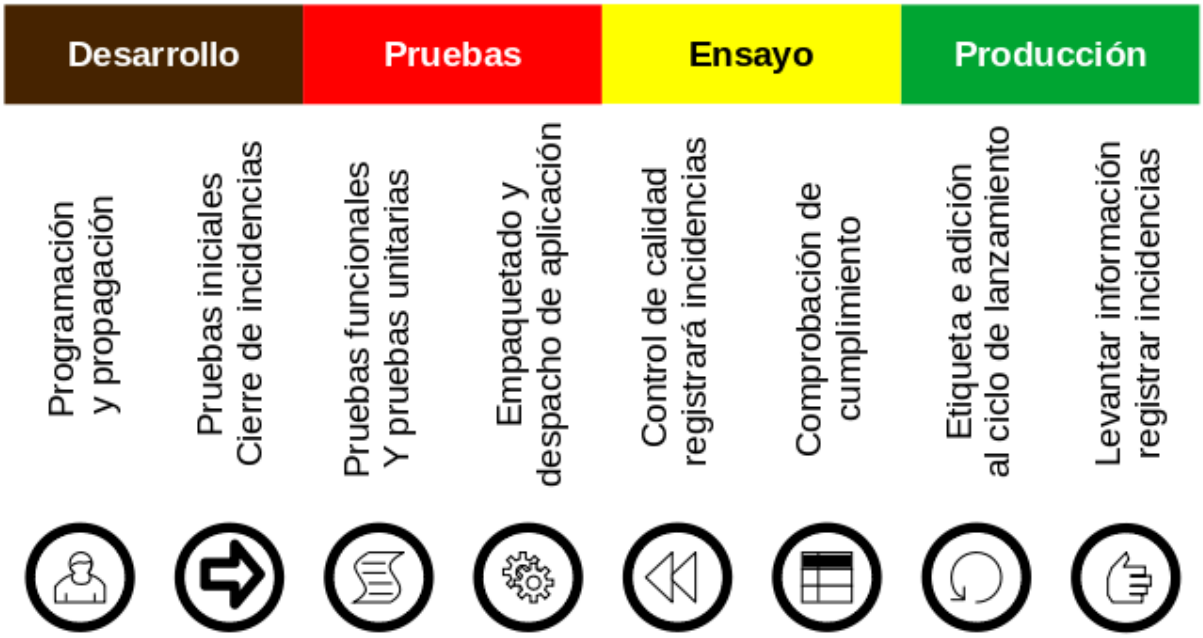
Rev v1.0.1 04.11.2022

## Informe de metodología

## Producción

El proceso de Producción del Software es un proceso dinámico, que debe entenderse como un movimiento o flujo de información contínua, el embotellamiento en las diferentes etapas de despacho o despliegue produce retrasos importantes; para evitar esto se han desarrollado a lo largo de los años estratégias de ingeniería que permiten agilizar y reaccionar rápidamente ante los problemas que surgen en el proceso, ya que de no poder observarlos estos se acumulan y obstruyen el proceso de producción.

El proceso de producción de Software en algunas organizaciones es más o menos complejo, dependiendo de los requerimientos que van surgiendo y la madurez del equipo, a continuación en el presente documento intentaré explicar un esquema básico y simple, que no es una norma general pero ayudará a estructurar los mecanismos de producción y agilizar los procesos.




La gráfica anterior muestra el flujo artefactos dentro del proceso de producción, esto es de izquierda a derecha desde el momento en el que se escriben las líneas que resuelven parte del problema, ahora bien, hay que tener claro en este punto que antes de todo lo descrito en este documento hay una serie de pasos previos, que no son parte del proceso de planificación, que es donde se describe la ingeniería y la arquitectura inicial, así como las tecnologías que se usarán en el proceso.

Para ello la gerencia de proyectos debería entregar a la dirección de desarrollo todos los detalles para que sean separados los problemas en pequeños fragmentos del problema

general, siguiendo el principio Rubik que indica que "*un problema complejo se resuelve más fácil fragmentando en problemas cada vez más pequeños y por consiguiente más simples.*"

De manera que cada incidencia o tarea conlleva a una sola responsabilidad y a la vez sencilla de explicar. Ahora bien, cargar estas incidencias y fragmentación del proyecto no debe ser responsabilidad del desarrollador, su responsabilidad es darle seguimiento, solucionarlo hasta cerrar cada incidencia.

Una vez que el desarrollador ha cerrado una incidencia debe sumar los cambios a la rama principal (master, main, trunk) o la rama principal de desarrollo (dev-master).

## Versionado semántico 2.0

El versionado es parte esencial en el desarrollo de software, en realidad no es posible llevar un control de versión semántico, ya que cada lanzamiento tendría que ser importante. En su lugar, tratamos de usar el número de la versión como un indicador de la cantidad de dolor que causará la actualización.

El formato de número de versión utilizado es <mayor>.<menor>.<parche>(-calificador)

Véase: semver.org

### Mejora la empatía

Es importante recordar que las actualizaciones no son una tarea agradable para la mayoría de los usuarios. Como equipo, siempre debemos brindar la mayor ayuda posible y tener empatía por los usuarios.

Siempre debemos considerar cuidadosamente las razones detrás de cualquier cambio que hagamos. A veces es mejor para nosotros asumir el dolor que de otro modo infligiríamos a los usuarios. Por ejemplo, a menudo elegimos admitir versiones anteriores de Java aunque nos gustaría usar funciones de lenguaje más nuevas nosotros mismos.

### Lanzamientos de parches

Los lanzamientos de parches deben ser reemplazos directos compatibles con API para los usuarios. Solo deben contener correcciones de errores.

### Versiones menores

Las versiones menores contienen nuevas características. En general, debería ser posible actualizar desde una versión secundaria anterior sin demasiado esfuerzo, siempre que la versión principal no tenga cambios y no se utilicen métodos obsoletos "Deprecated".

## Versiones mayores (principales)

Las versiones principales están reservadas para grandes cambios de última hora. Se espera que los usuarios tengan que trabajar un poco al actualizar las versiones principales.

Esto es porque la versión mayor se utiliza para hacer cambios que no son compatibles con el API.

## Calificadores

Los calificadores son parámetros opcionales que definen la versión actual, algunos ejemplos podrían ser:

- RELEASE
- SNAPSHOT
- tiger
- alpha
- BUILD20220719064923

### Releases

Representa una versión estable, única e inmutable, esto quiere decir que este artefacto no va a cambiar.

### Snapshots

Se refiere a una versión del proyecto que está en tiempo de desarrollo. Una especie de WORK-IN-PROGRESS o "Under Development". Parecido al concepto de Build en contraposición al de Release. En maven la convención es agregar el sufijo "-SNAPSHOT". Todo lo que antecede a este prefijo realmente no le importa a maven, puede tener letras, números, lo que sea (aunque ya veremos que también nos viene bien tener una convención). Ejemplo:

- 1.0-SNAPSHOT: se lee cómo "la que va a ser la versión 1.0"
- 1.0.1-SNAPSHOT: futura versión 1.0.1
- 1.0-alpha01-SNAPSHOT: lo que va a ser la versión 1.0-alpha01 etc.

Se trata de una captura instantánea, y es que a veces necesitamos crear una versión instantánea de las fuentes actuales considerando la fotografía instantánea para tener una idea.

Una consecuencia de esto es que las versiones snapshots son mutables, es decir que van evolucionando y cambiando. Si bajamos un snapshot hoy, quizás no sea igual al de ayer, o la semana pasada. Es ideal durante las etapas de desarrollo, pero no se utiliza para lanzamientos.

### Builds

Este es un calificador mucho más sencillo de comprender, se trata de una marca de compilación que contiene el timestamp del momento de empacarse, esto resulta muy útil para lanzamientos beta o inestables ya que contínuamente hay que reemplazarlos con su misma versión corregida por ejemplo 2.0.3-beta-build20220719064923 y 2.0.3-beta-build20220718083314 claramente se puede distinguir que aunque se trata de la misma versión ambos son paquetes diferentes.

## Política de depreciación

Las clases y los métodos en desuso deben anotarse @Deprecated de manera que sea claro que en las próximas versiones ya no estará disponible.

## Nuevas funciones frente a errores

Las nuevas funciones deben apuntar a la próxima versión menor y no deben incluirse en las versiones de parches. Los errores deben dirigirse a la versión compatible más antigua, a menos que sean particularmente riesgosos o difíciles de corregir en una rama más antigua, en algunos casos se puede usar la marca <mayor>-<menor> para dar seguimiento global a una versión, por ejemplo v2.0, v2.3.

## Confirmar mensajes

Los mensajes de confirmación son una herramienta vital cuando se trata de depurar una regresión. Es natural ser descuidado al crear mensajes de confirmación, sin embargo cuando se trabaja en equipo y siempre que sea posible los mensajes de confirmación descriptivos, lo recomendable es seguir estas 7 reglas para los mensajes de confirmación:

1. Separe el asunto del cuerpo con una línea en blanco
2. Límite la línea de asunto a 72 caracteres (use más corta si es posible)
3. Poner en mayúscula la línea de asunto
4. No termine la línea de asunto con un punto
5. Use el estado de ánimo imperativo en la línea de asunto
6. Envuelva el cuerpo en 72 caracteres
7. Usa el cuerpo para explicar qué y por qué versus cómo

Además, casi todos los mensajes de confirmación incluyen una referencia a un problema en el Seguimiento de incidencias o una solicitud de extracción (pull request). Hacemos referencia a problemas usando: "See", "Fixes" o "Closes" luego #NNNN. Por ejemplo:

*Permitir resultados opcionales de ConfigDataLocationResolver*

*Actualice `ConfigData` para que señale si se considera opcional. Este actualización permite que `ConfigDataLocationResolvers` devuelva resultados que comportarse de la misma manera que `opcional:` ubicaciones prefijadas sin el propio usuario necesita prefijar la cadena de ubicación. Cierra #322*

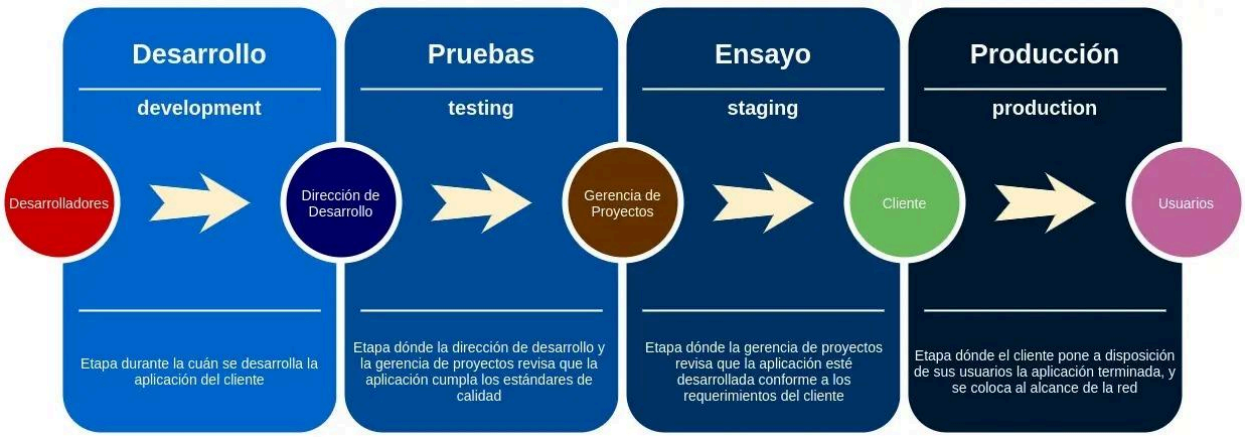
Vale la pena leer los siguientes blogs sobre el tema:

https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html

https://chris.beams.io/posts/git-commit/

## Entornos de Implementación

Queda sostenido que la planificación, el versionado y la comunicación contínua permiten un manejo de soluciones más eficiente, por lo tanto el desarrollo se pondrá a disposición del equipo de Ensayo (Staging) de forma instantánea mediante herramientas de integración contínua y distribución contínua (CI/CD), (Se explica en otro documento).




### Desarrollo (development)

Etapa durante la cuál se desarrolla la aplicación, el entorno de desarrollo suele estar al alcance directo del desarrollador, esto es posiblemente en su propio equipo o en un servidor en su oficina, ya que el desarrollador debe haber realizado las pruebas de primera mano antes de realizar un despliegue a testing. En algunos casos el desarrollador requiere Certificados SSL o conexiones VPN para lograr simular un entorno de producción, en esta etapa podría trabajar con entornos de prueba y datos de relleno, pero esto deben servirse cerca y no desde los servidores de despliegue o se compromete los tiempos de desarrollo por asuntos de latencia, velocidad o problemas de disponibilidad de los datos. Para ello lo

natural es entregar especificaciones claras de las interfaces de datos que maneja el entorno de ensayo y producción de manera que este pueda crearse una caja de arena para prácticas (Sandbox) mientras desarrolla el código fuente.

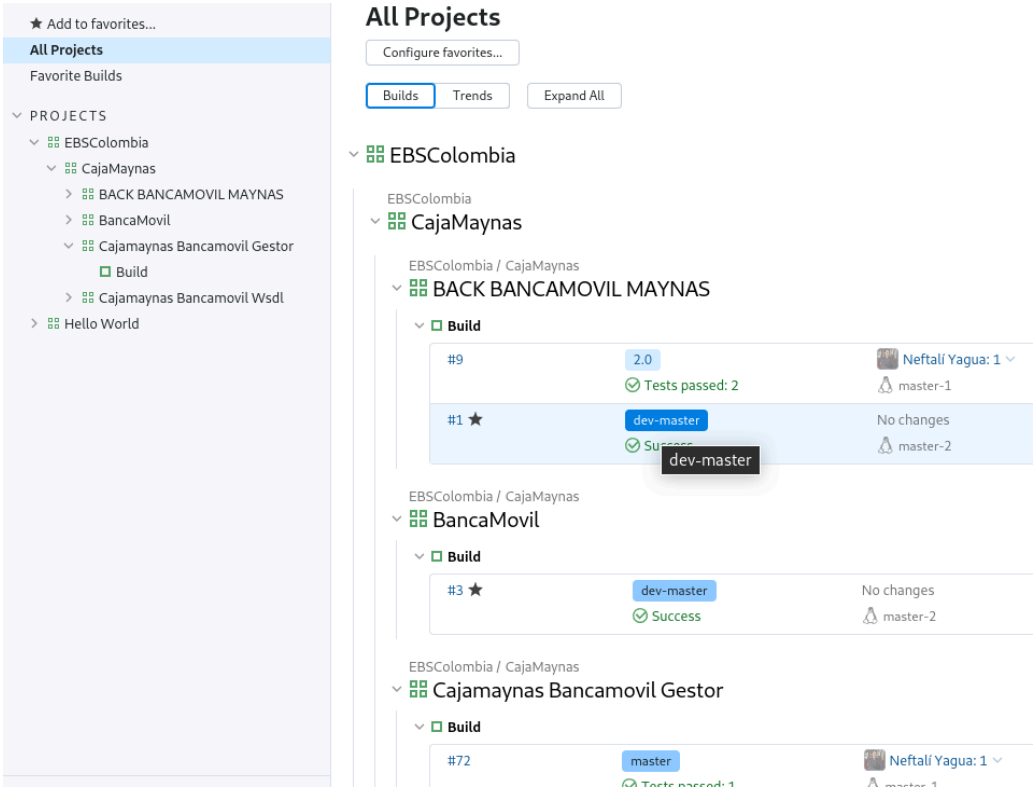
### Desarrolladores

Lo natural es que los desarrolladores no deberían preocuparse por asuntos de infraestructura, en su lugar debería haber un canal donde pudieran solicitar la infraestructura que requieran.

Establecer canales de comunicación entre los desarrolladores es un elemento clave en especial durante esta etapa, y estos canales no deben ser las herramientas tradicionales, sino herramientas especializadas para el manejo de equipos.

## Pruebas (testing)

La etapa de pruebas funcionales y unitarias, no es un proceso de prueba física de la aplicación, sino de pruebas técnicas al código, que permita examinar si los parámetros de calidad en relación a cada función que debe hacer el Software se pueda comprobar de forma automática, esto es porque en el proceso de desarrollo de Software los desarrolladores contínuamente están compartiendo cambios y fusionando ramas, lo que requiere en primer lugar que cada solicitud de integración sea revisada exhaustivamente antes de unirlo a la rama principal (o de desarrollo, en algunos casos se usan dos ramas principales). Esto es un proceso prácticamente automático, por lo tanto no debe requerir ningún tipo de intervención, salvo los cambios escritos por el desarrollador al crear las pruebas funcionales y unitarias, es común confundir estas pruebas con pruebas integrales al software ejecutadas por un operador, pero ese tipo de pruebas corresponden a la etapa de ensayo ya que las pruebas funcionales corresponden a la edición del código fuente y ejecución en el servidor, a continuación un ejemplo:



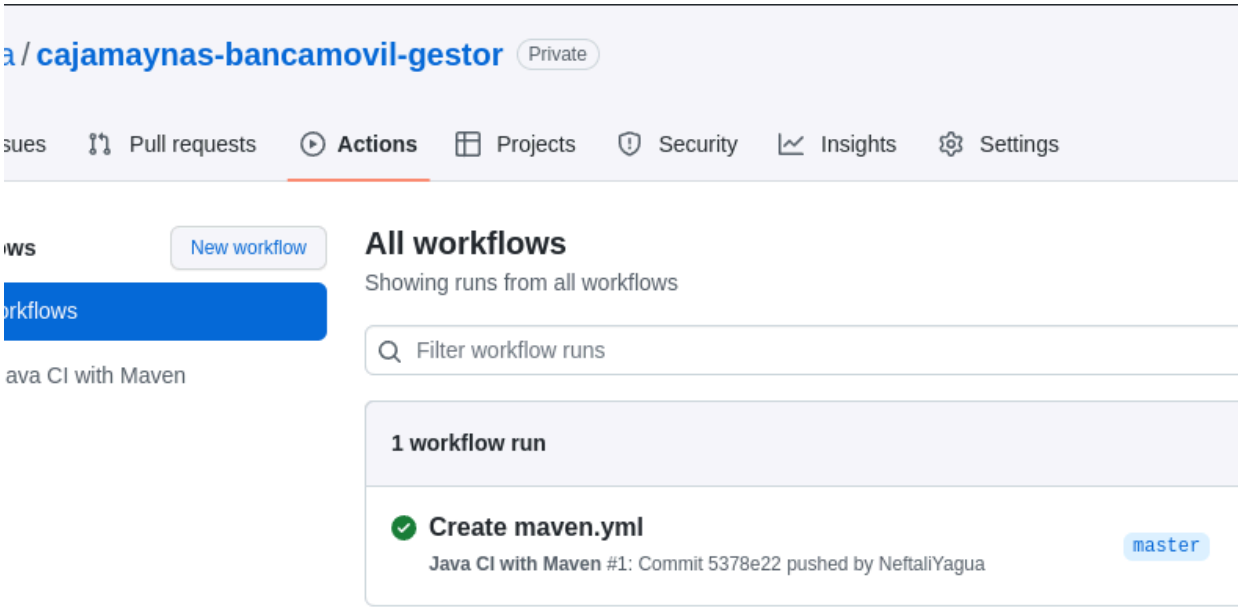

El equipo de desarrollo es decir, quien envía la fuente y el resto del equipo son los encargados de monitorear esta etapa es decir, que no es responsabilidad solamente del desarrollador, sino de los demás integrantes, ya que enviar código fuente para ser examinado en el servidor de testing debería ser un paso sencillo, cabe destacar que es totalmente válido configurar el servidor para hacer el despliegue a la siguiente etapa (staging) o fusionar el código fuente.

## Dirección de desarrollo

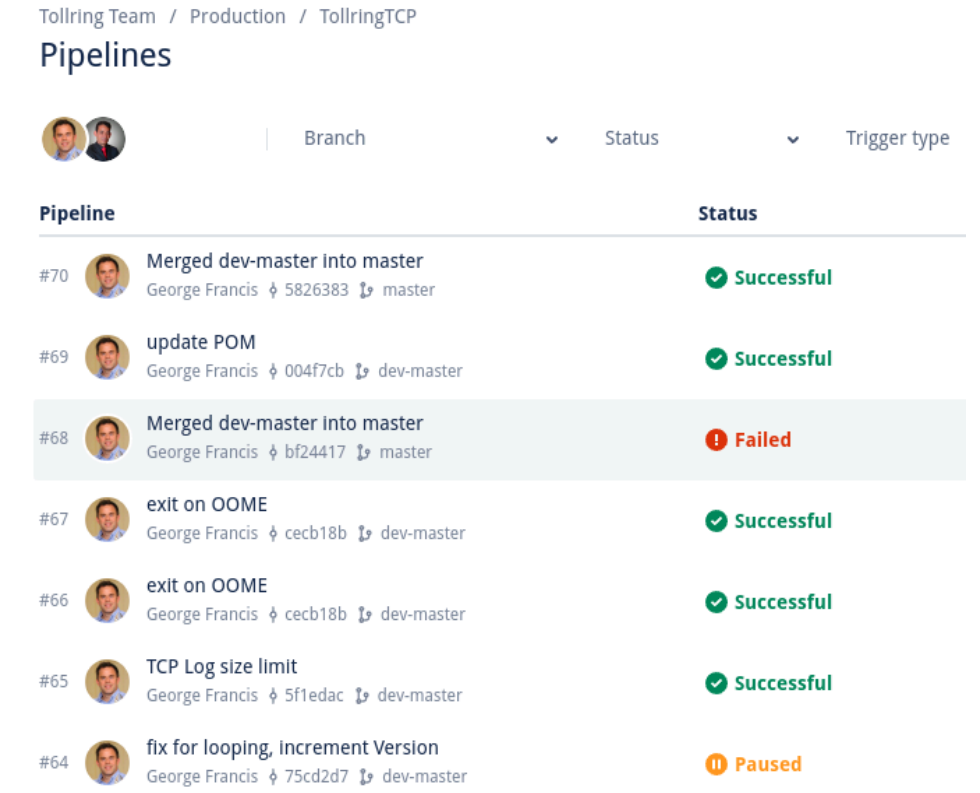
La dirección de desarrollo debería proveer a los desarrolladores la configuración de canalización de pruebas funcionales o en su defecto proveer un servidor para testing o como mínimo, indicar las pautas que conciernen a esta etapa.

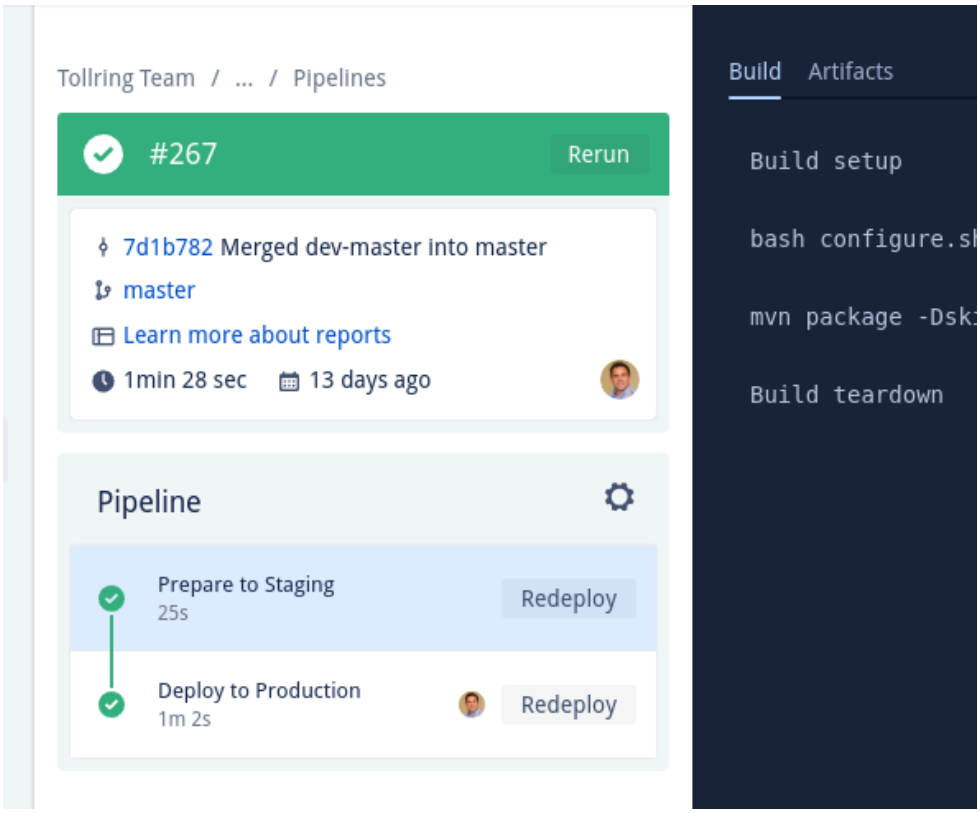
Lo ideal es tener un servidor dedicado a las pruebas sin embargo los servicios de control de versiones suelen implementar herramientas para este tipo de pruebas.

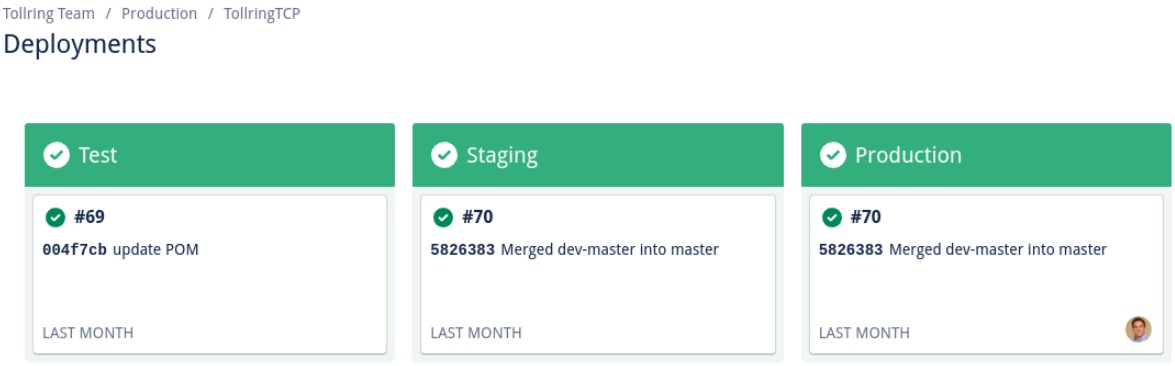
A continuación un ejemplo de actions en github:


a / cajamaynas-bancamovil-gestor
Private
sues
Pull requests
Actions
Projects
Security
Insights
Settings
ows
New workflow
orkflows
ava CI with Maven
All workflows
Showing runs from all workflows
Filter workflow runs
1 workflow run
Create maven.yml
Java CI with Maven #1: Commit 5378e22 pushed by NeftaliYagua
master

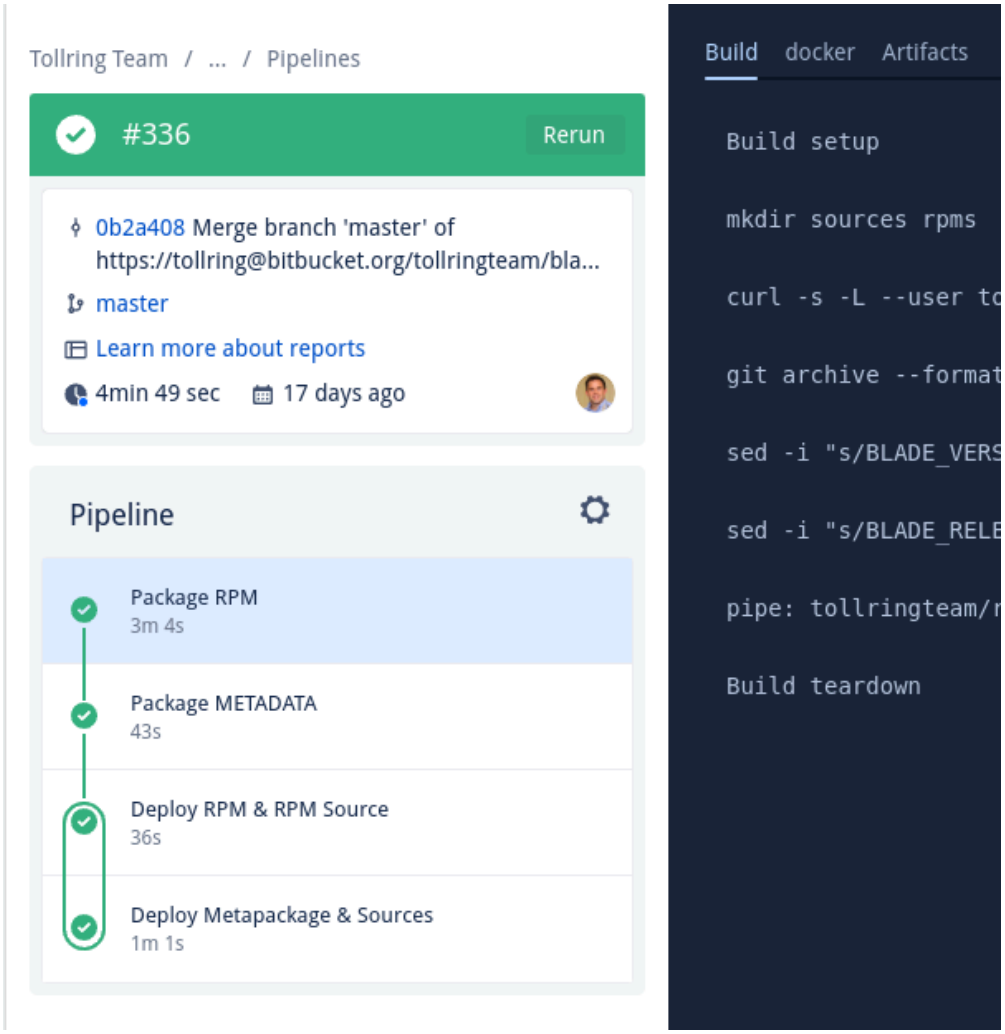
A continuación pueden verse algunos ejemplos en bitbucket:



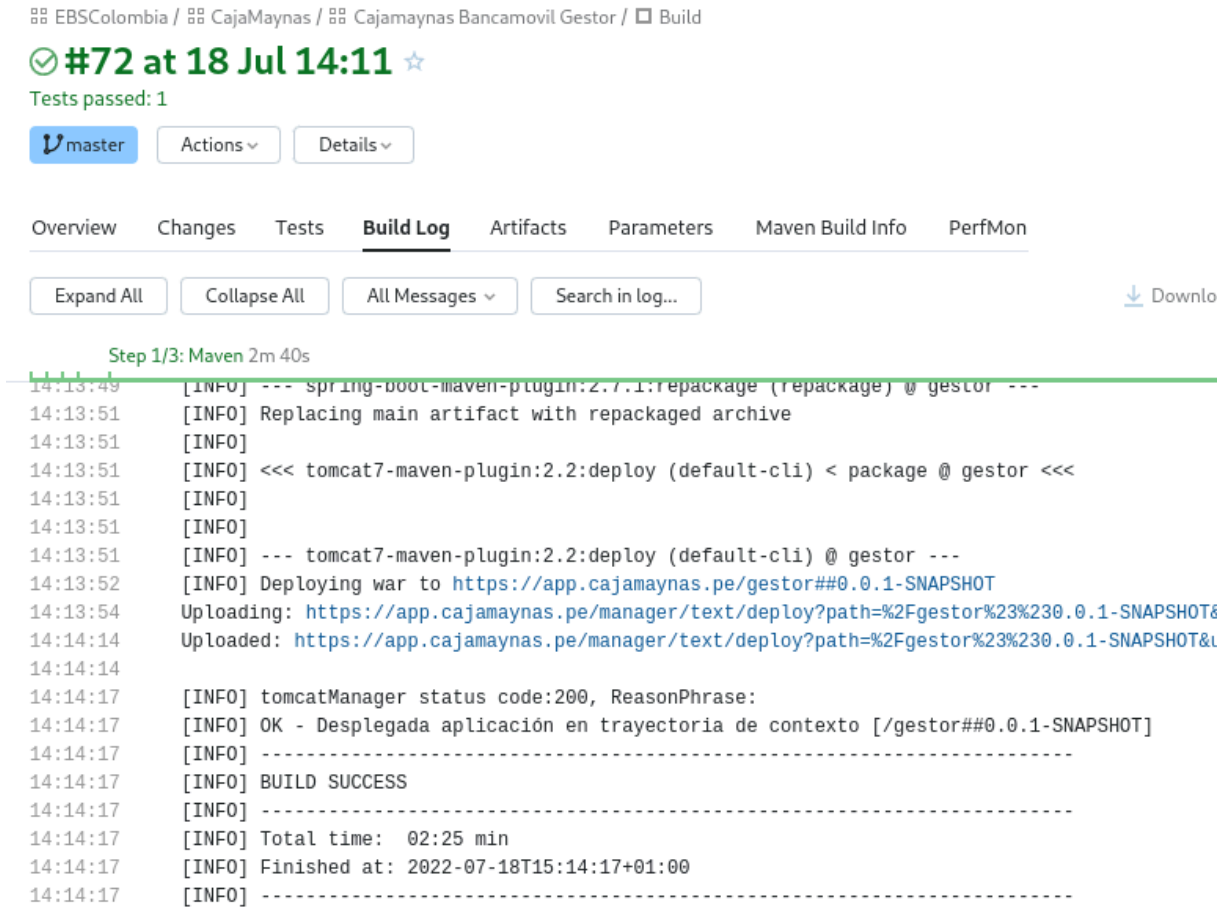






Una configuración de pruebas adecuada podría desplegar automáticamente el Software al entorno de ensayo (Que no es el entorno de producción ni el de desarrollo, es importante observar siempre esta diferenciación). A continuación otro ejemplo que muestra cómo se empaqueta un RPM  a través de la canalización en bitbucket para enviarlo al repositorio de ensayo (staging) para posteriormente ser probado por el equipo de calidad:



Esto también puede hacerse desde la infraestructura propia, por ejemplo a continuación podemos ver como el servidor de pruebas como el software TeamCity empaqueta el código fuente y lo despliega de forma automática al servidor de tomcat para que pueda ser examinado por el equipo de Calidad en un entorno de ensayo.



**Gestor de Aplicaciones W**

| Mensaje: | OK |
|---|---|

| *Gestor* | | |
|---|---|---|
| Listar Aplicaciones | Ayuda HTML de Gestor | |

| *Aplicaciones* | | | |
|---|---|---|---|
| **Ruta** | **Versión** | **Nombre a Mostrar** | **Ejecutándose** |
| /api/v1 | 2.0 | CajaMaynas Rest API gateway | true |
| /gestor | 0.0.1-SNAPSHOT | | true |
| /host-manager | *Ninguno especificado* | Tomcat Host Manager Application | true |
| /manager | *Ninguno especificado* | Tomcat Manager Application | true |

Es importante tener en cuenta que el equipo de Calidad es quién debe estar pendiente de recibir los despliegues que son despachados del entorno de pruebas a ensayo, ya que los desarrolladores no deberían tener que ocuparse de esto, en su lugar mantenerse enfocados en su entorno de desarrollo desplegando nuevos cambios documentados en el servicio de seguimiento de incidencias, no observar este hecho podría producir retrasos considerable en las etapas de desarrollo, ya que sobre ellos pesa la garantía de que todos los paquetes que llegan al entorno de ensayo han pasado la comprobación de pruebas funcionales y unitarias configuradas en el código fuente.

## Ensayo (staging)

En el entorno de ensayo, el equipo de calidad puede realizar una revisión de las funcionalidades que están pendientes y que no están registradas en el control de incidencias, y añadir uno a uno al control de incidencias, que es la herramienta adecuada para garantizar que los objetivos se cumplan, cabe destacar que el flujo de la aplicación a través de los entornos no incrementa la versión en ninguna manera excepto cuando la aplicación pasa de la etapa de ensayo a producción.

### Gerencia de proyectos

Dentro de la gerencia de proyectos se examina con detenimiento que se cumplan todos y cada uno de los requerimientos del cliente, y son quiénes inician las incidencias de tareas y especifican el ciclo o versión a la que pertenecen, de manera que los desarrolladores sólo deben seguir la dirección establecida.

### El Cliente

Después de que el equipo de producción está satisfecho con el cumplimiento de los requerimientos es cuando procede a discutir con el cliente y hacer presentaciones, en este punto el Software ya debe haber superado todas las pruebas exhaustivas de manera que el cliente tiene en la mano la garantía de un Software  funcional y probado.

## Producción (production)

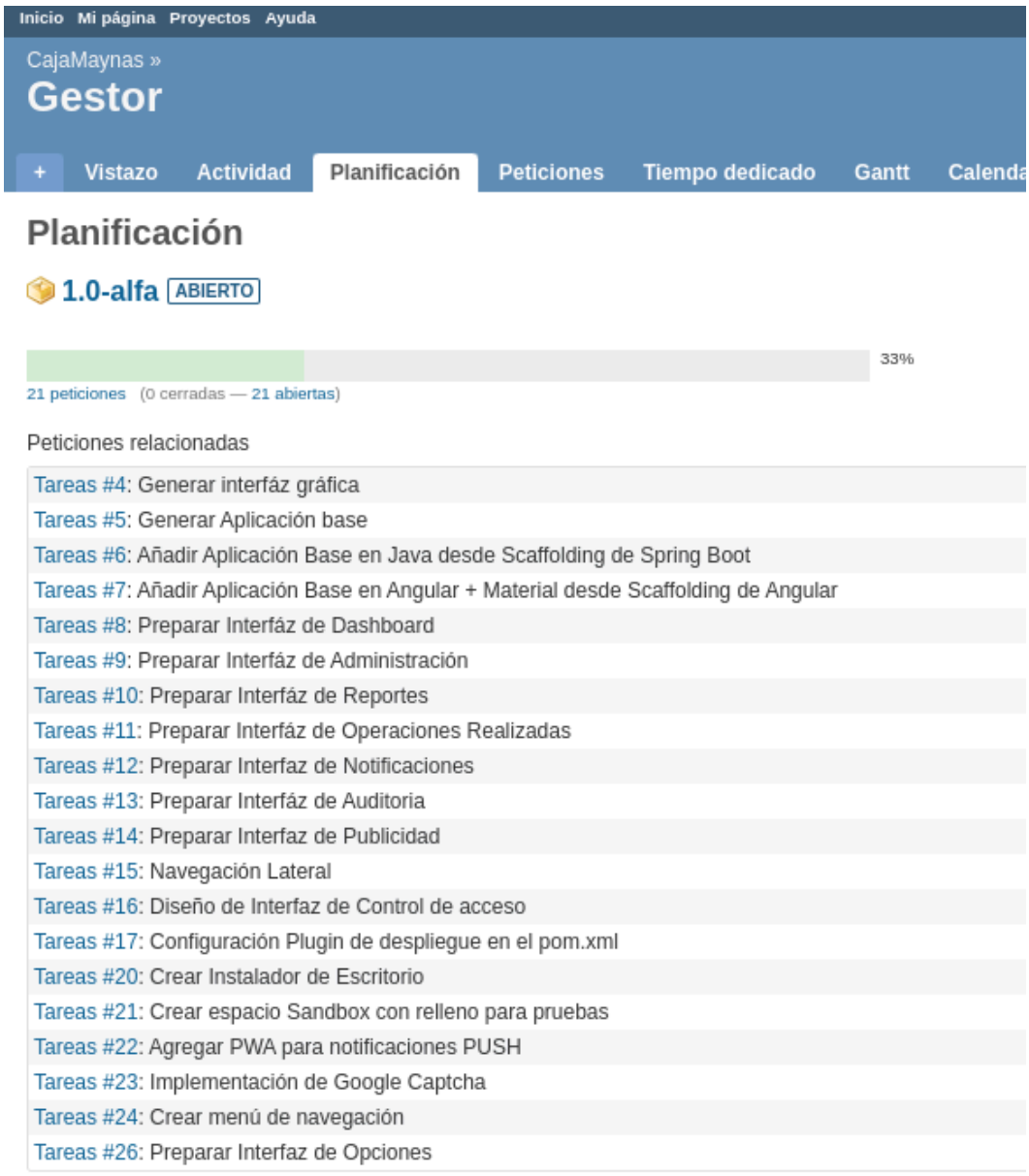
Es el equipo de Calidad quién determina cuándo una versión está terminada y prosigue iniciar el ciclo de lanzamiento de la versión actual o a continuarlo en caso de que ya esté iniciado previamente, entendemos que una misma versión tiene varios lanzamientos, y no se crea la etiqueta tag hasta que la versión está terminada, entendiendo que los tags no se modifican de manera que determina un estado constante del código fuente que le compone, en este sentido se entiende que los tags son para versiones cerradas de modificaciones, y las ramas para versiones abiertas a modificaciones esto conforme al principio Open/Closed que conforma los principios SOLID de Robert C. Martin.

### Usuarios finales

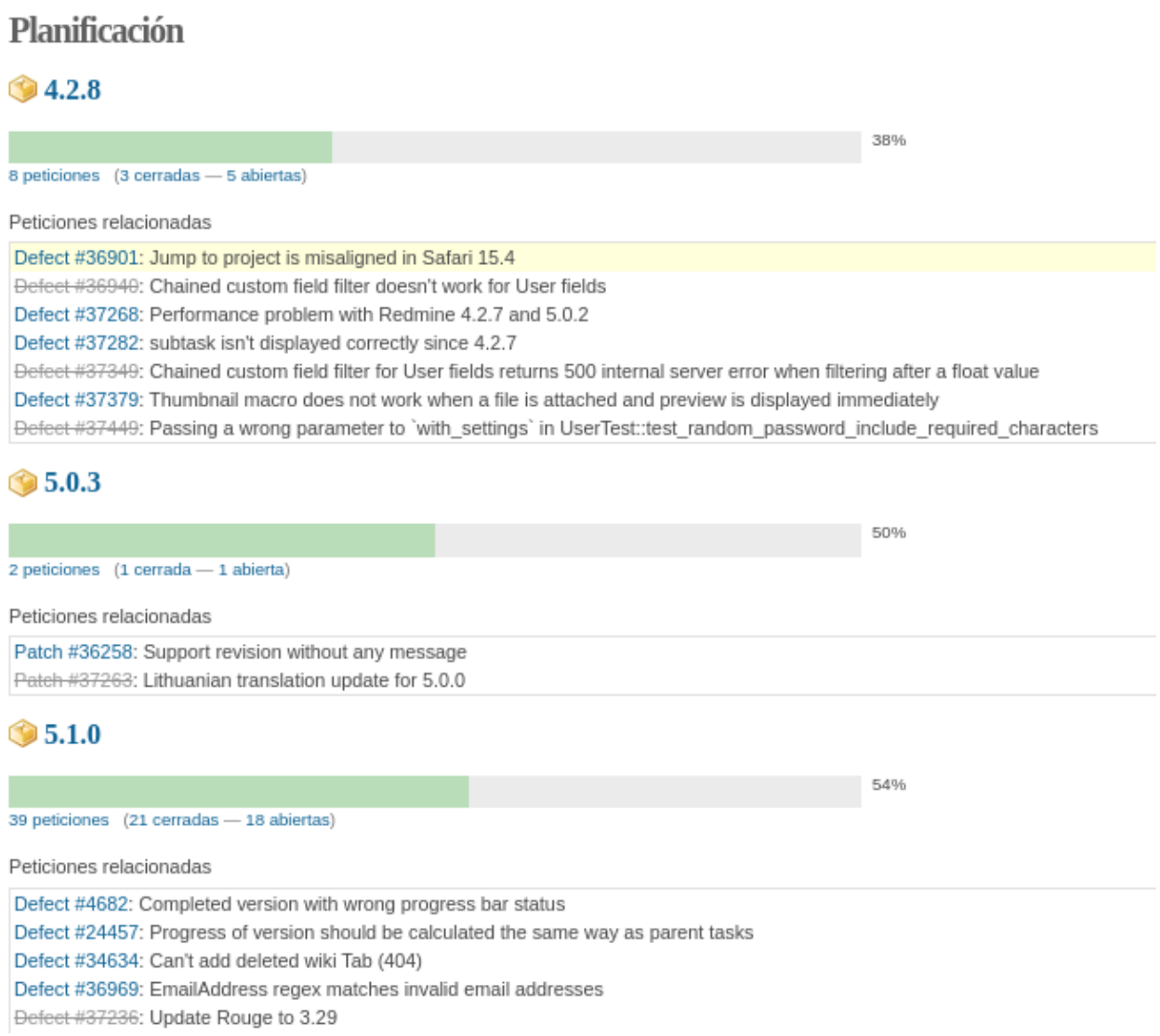
Es importante establecer canales de comunicación con los usuarios finales, y especificar con claridad el ciclo de lanzamiento del Software, de manera que si una versión es beta el usuario sabrá que se trata de una versión inestable donde se esperan problemas.

## Seguimiento de incidencias

Es el equipo de calidad los encargados de garantizar que las pruebas operativas (No confundir con unitarias y funcionales que esas si las hacen los desarrolladores en el código) sean revisadas arduamente y que una vez que se presentan al cliente o usuario final se considere que no hay errores conocidos sin registrar, pues el equipo de calidad debería crear un ticket de incidencia con cada falla que encuentre de manera que en el desarrollo se pueda llevar un seguimiento y corrección.



Todos los errores encontrados en una versión cerrada se añaden en la cola para la siguiente versión de parches, todas las modificaciones que incrementan las características del Software pero que continúan siendo compatibles con el API se añaden a la próxima versión menor, y todas las modificaciones o cambios que no son compatibles se añaden a la cola de incidencias de la siguiente versión mayor, por ejemplo:



Planificación

4.2.8

38%

8 peticiones (3 cerradas — 5 abiertas)

Peticiones relacionadas

Defect #36901: Jump to project is misaligned in Safari 15.4
~~Defect #36940~~: Chained custom field filter doesn't work for User fields
Defect #37268: Performance problem with Redmine 4.2.7 and 5.0.2
Defect #37282: subtask isn't displayed correctly since 4.2.7
~~Defect #37349~~: Chained custom field filter for User fields returns 500 internal server error when filtering after a float value
Defect #37379: Thumbnail macro does not work when a file is attached and preview is displayed immediately
~~Defect #37449~~: Passing a wrong parameter to `with_settings` in UserTest::test_random_password_include_required_characters

5.0.3

50%

2 peticiones (1 cerrada — 1 abierta)

Peticiones relacionadas

Patch #36258: Support revision without any message
~~Patch #37263~~: Lithuanian translation update for 5.0.0

5.1.0

54%

39 peticiones (21 cerradas — 18 abiertas)

Peticiones relacionadas

Defect #4682: Completed version with wrong progress bar status
Defect #24457: Progress of version should be calculated the same way as parent tasks
Defect #34634: Can't add deleted wiki Tab (404)
Defect #36969: EmailAddress regex matches invalid email addresses
~~Defect #37236~~: Update Rouge to 3.29

## Ciclo de vida del lanzamiento de software

Cabe destacar que para un correcto desarrollo del Software es necesario respetar cabalmente los ciclos de lanzamiento, que es tan importante como los entornos de implementación observados anteriormente. Y entender que hay un espacio durante el desarrollo dónde ni el cliente final ni el usuario final participan, y esto es porque en las etapas de depuración del código es absolutamente normal encontrarse con errores, y esto no debería conocerlo el cliente hasta ser corregidos.

### Alfa. Etapa inicial

La etapa inicial de la versión debe tener todas las características del requerimiento del cliente, incluye todo lo que se necesita para funcionar, es lo que se espera, sin embargo todavía debe someterse a una serie de pruebas exhaustivas que determinan cuáles cambios deben hacerse, para alcanzar con exactitud el requerimiento del cliente; y en especial para resolver los problemas planteados en la primera exploración de las necesidades del proyecto.

### Beta. Etapa de pruebas

Esta etapa de la versión procura hacer una exploración profunda de la aplicación, para conocer posibles fallas, repararlas y corregir alguna que su uso pueda ocasionar en los datos, es una aplicación que bién se puede usar definitiva, esta será liberada de su responsabilidad ya que encontrar errores en ella es lo que se espera, etapa imprescindible para alcanzar el máximo nivel de calidad.

### RC. Etapa Candidata a definitiva (Release Candidate)

Esta etapa de la versión es prácticamente el producto terminado, listo para usarse sin embargo se mantiene bajo estricta observación, debe estar funcionando correctamente en todo sentido, atentos a recuperar todos los datos necesarios para mejorar la aplicación.

### Final. Etapa definitiva

Esta etapa de la versión se alcanza cuando se entiende por terminada la aplicación, lista para usar y puesta en marcha definitiva al público en producción.

## Conclusión

En primer lugar, dejar claro que los principios normas y técnicas descritas en el presente documento no son una ley absoluta acerca de cómo se deben organizar los entornos, versiones y ciclos de lanzamientos, históricamente se ha comprobado que son parte de las buenas prácticas que nos ayudan a organizar el desarrollo y el versionado, pero muy en especial el mantenimiento rápido de cambios. Este documento es una primera versión que puede mejorarse aún más, en la medida que los equipos de trabajo tienen principios claros sobre cómo se marcan los flujos de trabajo, es más sencillo hacer el mantenimiento de desarrollo de Software, actualmente hay tecnologías mucho más avanzadas para esto, pero antes de ir allí hay que empujar al equipo a dominar principios como estos.