FACULTAD DE INGENIERÍA

Carrera de Ingeniería de Sistemas Computacionales

"INFLUENCIA DE UN CONTINUOUS DELIVERY PIPELINE EN EL PROCESO DE DESPLIEGUE DE REQUERIMIENTOS DE UN SOFTWARE"

Tesis para optar el título profesional de:

Ingeniero de Sistemas Computacionales

Autor:
Richard Manuel Paredes Atencio

Asesor:
Mg. Ing. Patricia Janet Uceda Martos

Cajamarca - Perú

2019

DEDICATORIA

A Dios por haberme dado la vida y por haber puesto en mi camino a aquellas personas que me apoyaron en toda mi carrera. A ti madre por haberme dado la vida, por tus consejos que me llevan a ser una mejor persona. A mi padre, por brindarme apoyo, por creer en mí y porque eres un ejemplo a seguir. A mis hermanos, por apoyarme y recordarme que debo esforzarme para conseguir mis sueños.

AGRADECIMIENTO

Agradezco a mi asesora de tesis Mg. Patricia Uceda Martos, por su paciencia leyendo y corrigiendo el desarrollo de esta investigación,

también por su apoyo y confianza en mi trabajo y su forma de guiar mis ideas en el desarrollo de esta investigación y en mi formación profesional, demostrando paciencia y deseo de compartir conocimientos.

Tabla de contenidos

DEDICATORIA **2**

AGRADECIMIENTO **3**

ÍNDICE DE TABLAS **6**

ÍNDICE DE FIGURAS **7**

CAPÍTULO I. INTRODUCCIÓN **10**
1.1. Realidad problemática 10
1.2. Formulación del problema 31
1.3. Limitaciones 31
1.4. Objetivos 32
1.5. Hipótesis 32

CAPÍTULO II. METODOLOGÍA **34**
2.1. Tipo de investigación 34
2.2. Materiales, instrumentos y métodos 34
2.3. Técnicas e instrumentos de recolección de datos 34
2.4. Procedimiento 35

CAPÍTULO III. RESULTADOS **36**
Objetivo Específico 1 36
Objetivo Específico 2 43
Objetivo Específico 3 50
Prueba contrastación de hipótesis con Wilcoxon 56

CAPÍTULO IV. DISCUSIÓN Y CONCLUSIONES **63**
4.1. Discusión 63
4.2. Conclusiones 64

REFERENCIAS **66**

ANEXOS **70**
Anexo 1 Operacionalización de variables 70
Anexo 2 Fichas de validación de instrumento 71
Anexo 3 Instrumento proceso de despliegue 74
Anexo 4 Confiabilidad del instrumento con Alpha de Cronbach 75
Anexo 5 Estadísticos Wilcoxon 76
Anexo 6 Requerimientos de prueba 81
Anexo 7 Documentos desarrollo Proyecto 83
Anexo 8 Instalación de Bamboo 92
Anexo 9 Instalación de SonarQube 96
Anexo 10 Configuración de análisis de calidad de código 98
Anexo 11 Configuración despliegue automatizado 104
Anexo 12 Caso de uso Subir cambios de código y realizar análisis de calidad de código 107
Anexo 13 Caso de uso Realizar despliegue automatizado 109

Anexo 14 Caso de uso Realizar rollback automatizado 111
Anexo 15 Caso de uso Realizar despliegue manual 114
Anexo 16 Descripción general del proyecto portal de empleo..... 116
Anexo 17 Elección de herramientas 127

ÍNDICE DE TABLAS

Tabla 1 Tareas kanban del desarrollo de la investigación 36
Tabla 2 Presupuesto inicial del proyecto 39
Tabla 3 Indicadores de calidad de software 43
Tabla 4 Ejemplo de requerimiento tipo cambio 43
Tabla 5 Resultado pre-test de calidad de software 44
Tabla 6 Resultado post-test de calidad de software 45
Tabla 7 Indicadores de agilidad y calidad del proceso de desarrollo 50
Tabla 8 Resultado pre-test de indicadores de agilidad y calidad del proceso de desarrollo 51
Tabla 9 Resultado post-test de indicadores de agilidad y calidad del proceso de desarrollo 52
Tabla 10 Requerimientos de prueba 81
Tabla 11 Interesados y colaboradores del proyecto 85
Tabla 12 Organización del proyecto 87
Tabla 13 Listado de requerimientos del proyecto 88

ÍNDICE DE FIGURAS

Figura 1. Procesamiento de datos 35
Figura 2. Tablero Kanban en Jira 38
Figura 3. Configuración WIP en Jira 38
Figura 4. Registro de tiempo de tarea en Jira 39
Figura 5. Diseño del continuous delivery pipeline 40
Figura 6. Comparación de número de errores de Pre test y Post test 47
Figura 7. Comparación de número de vulnerabilidades de Pre test y Post test 47
Figura 8. Comparación de número de code smells de Pre test y Post test 48
Figura 9. Comparación de tiempo de deuda técnica en días de Pre test y Post test 48
Figura 10. Comparación de porcentaje de código duplicado de Pre test y Post test 49
Figura 11. Comparación de bloques duplicados de Pre test y Post test 49
Figura 12. Comparación de tiempo de solución de requerimiento en minutos de Pre test y Post test 54
Figura 13. Comparación de tiempo de integración de código en segundos de Pre test y Post test 55
Figura 14. Comparación de tiempo que toma realizar despliegue en minutos de Pre test y Post test 56
Figura 15. Confiabilidad del instrumento con Alpha de Cronbach 75
Figura 16. Prueba Wilcoxon a indicador número de errores 76
Figura 17. Prueba Wilcoxon a indicador número de vulnerabilidaes 76
Figura 18. Prueba Wilcoxon a indicador número de code smells 77
Figura 19. Prueba Wilcoxon a indicador tiempo de deuda técnica 77
Figura 20. Prueba Wilcoxon a indicador porcentaje de código duplicado 78
Figura 21. Prueba Wilcoxon a indicador número de bloques duplicados 78
Figura 22. Prueba Wilcoxon a indicador tiempo de solución de requerimiento en minutos 79
Figura 23. Prueba Wilcoxon a indicador tiempo de integración de código en segundos 79
Figura 24. Prueba Wilcoxon a indicador tiempo que toma realizar despliegue en minutos 80
Figura 25. Etapas para desarrollar el proyecto 86
Figura 26. Continuous delivery pipeline implementado 90
Figura 27. Ejemplo configuración Java 93
Figura 28. Crear nuevo plan de integración 100
Figura 29. Configuración de tareas de plan de integración 101
Figura 30. Configuración de tarea análisis de calidad de código en plan de integración 102
Figura 31. Archivo de configuración SonarQube 103
Figura 32. Crear plan de despliegue 104
Figura 33. Script KillProcessPort8000 105
Figura 34. Script RemoveAndCreateDir 105
Figura 35. Script RemovePythonlog.sh 106
Figura 36. Script RestartNginx ActivateEnvStartGunicorn 106
Figura 37. Iniciar despliegue 109
Figura 38. Resultado despliegue 110
Figura 39. Interfaz despliegue 111
Figura 40. Iniciar rollback 112
Figura 41. Resultado rollback 112
Figura 42. Casos de uso del proyecto portal de empleo 117
Figura 43. Diagrama de flujo iniciar sesión 118
Figura 44. Diagrama de flujo listar ofertas 118
Figura 45. Diagrama de flujo registrar oferta 119
Figura 46. Diagrama de flujo editar oferta 119
Figura 47. Diagrama de flujo ver detalle de oferta 120
Figura 48. Diagrama de flujo postular a oferta 120
Figura 49. Diagrama de flujo listar postulaciones 120
Figura 50. Diagrama de flujo ver curriculum 121
Figura 51. Diagrama de flujo listar curriculum 121
Figura 52. Diagrama de flujo editar curriculum 121
Figura 53. Diagrama de flujo registrar curriculum 122

Figura 54. Diagrama de clases iniciar sesión 122
Figura 55. Diagrama de clases ofertas 123
Figura 56. Diagrama de clases postulación 123
Figura 57. Diagrama de clases curriculum 124
Figura 58. Arquitectura MVC 125
Figura 59. Comparación Arquitectura MTV y MVC 125
Figura 60. Diagrama de despliegue 126

RESUMEN

En la actualidad muchas empresas que desarrollan software buscan mejorar su productividad intentando disminuir el tiempo de ciclo de desarrollo de software sin afectar la calidad de código; sin embargo, no adoptan nuevas tecnologías que ayudan a cumplir este objetivo, aunque esto depende de la cultura del equipo desarrollador. En tal sentido, la presente investigación intenta determinar la influencia de un continuous delivery pipeline en el proceso de despliegue de requerimientos de un software y así aportar con una posible solución a los tiempos extensos que toma hacer despliegues manuales y a controlar la calidad de código, diseñando un continuous delivery pipeline, la cual permitirá realizar análisis de calidad de código con el software SonarQube, realizar despliegues y rollback automáticamente con el software Bamboo; para esto se realizaron pruebas con un software basado en Python al que llamamos proyecto portal de empleo, para conocer si es posible que se use este pipeline en la solución de requerimientos, se resolvieron 20 requerimientos con dicho pipeline y de la forma en la que se resuelven actualmente, es decir con despliegues manuales y sin análisis de calidad de código. Al finalizar la investigación se demuestra que el continuous delivey pipeline influye positivamente en el proceso de despliegue de requerimientos, ya que han obtenido una mejora de calidad de código y de tiempo de despliegue pasando de realizarse en 8.75 minutos a 0.72 minutos en promedio.

Palabras clave: proceso de despliegue, calidad de código, continuous delivery pipeline.

CAPÍTULO I. INTRODUCCIÓN

1.1. Realidad problemática

Según Perera, Bandara y Indika (2016), en las dos últimas décadas el software ha llegado a ser una parte esencial de diversos negocios, los tipos de software son una variedad de aplicaciones de escritorio, web y móviles, debido a su compleja naturaleza los requerimientos cambian rápidamente, esto ha sido un desafío para los equipos de desarrollo para producir software que satisfaga las necesidades de los clientes con las funcionalidades esperadas, la calidad del software y el despliegue a tiempo. Cabe resaltar que compañías grandes como Amazon y Netflix son desplegadas miles de veces al día. Otro caso es el que mencionan Meng, Wegman, Zhang, Chen y Chafle (2017), quienes afirman que en el año 2011, los ingenieros de Facebook hicieron commit de su código a sus sistemas de control de versiones hasta 500 veces por día y desplegaron estas actualizaciones a servidores semanalmente, diariamente e incluso cada hora.

De acuerdo a Meng, Wegman, Zhang, Chen, y Chafle (2017), la necesidad de agilidad ha resultado en el desarrollo y despliegue continuo de frecuentes actualizaciones pequeñas en los ambientes de producción de tecnología de la información (TI), sin embargo, las aplicaciones en constante cambio y sus entornos de TI desafían los enfoques de solución de problemas de TI existentes, los cuales con frecuencia tienen una gran dependencia del conocimiento humano y su experiencia; estas implican operaciones manuales. Resaltando que un despliegue manual es una operación, Savor, et al. (2017), afirman que los despliegues manuales incrementan el riesgo de fallos de infraestructura, esto provocaría que no se desplieguen o se desplieguen con errores, contradiciendo a Farley y Humble (2010), ya que ellos dicen que la rapidez es esencial porque hay una oportunidad de costo asociado con no hacer el despliegue de un

software. De acuerdo a la International Data Corporation (IDC), para las organizaciones con ingresos de al menos $ 1,39 mil millones anuales, el costo promedio por hora de un fallo de infraestructura es de aproximadamente $ 100 000. En un caso particular según Savor, et al. (2017), en el año 2013, Amazon perdió $ 66 240 por minuto en ventas durante un corte de servicio debido a un error de infraestructura.

Farley y Humble (2010) afirman que, en muchos proyectos de software, el proceso de despliegue es un proceso manual intenso, la razón para el nerviosismo debe ser clara, ya que hay bastante que puede ir mal. Si cualquier paso no es perfectamente ejecutado, la aplicación no va a funcionar apropiadamente. Cuando los despliegues no son completamente automatizados, los errores van a ocurrir cada vez que son realizados. Testear un manual de procesos de despliegues usualmente consume mucho tiempo, es caro y no hay garantía que la documentación haya sido seguida, en cambio un proceso de despliegue automatizado es barato, fácil de testear y de hacer auditorías. El proceso de despliegue automatizado debe ser usado por todos, y esta debe ser la única forma en la cual el software es desplegado. Cabe recalcar que de acuerdo con Meng, Wegman, Zhang, Chen, y Chafle (2017), el 80% de problemas de funcionamiento de aplicaciones han sido causadas por falta de comunicación o falta de entendimiento de manuales de procesos de despliegues; el problema se agrava, ya que según afirman Savor, et al. (2017), más del 85% de los problemas de funcionamiento y configuración no proporcionan ni siquiera mensajes engañosos que permitan entender el error de infraestructura, lo que provoca mucho esfuerzo para diagnosticar problemas.

Farley y Humble (2010), afirman que el despliegue de un software puede y debería ser de bajo riesgo, frecuente, barato, rápido y un proceso predecible, incluso Roche (2013), manifiesta que el despliegue debe ser indetectable para los usuarios, algunos

de los cuales tendrán la versión del software cambiado en mitad de sesión. Además, Mahanta, Adige, Pole, y Rajkumar (2016), aseguran que para las organizaciones es altamente recomendable que el tiempo para responder a los requerimientos de los clientes debe ser mínimo y así se logre cumplir los Service Level Agreement (SLAs) requeridos. De esta manera, según afirman Shahina, Babara, y Zhub (2015), teniendo frecuencia de despliegues y releases confiables, se logrará mejorar la satisfacción de los clientes y la calidad del producto software, también se lograría cumplir lo que confirman Savor, et al. (2017), los desarrolladores prefieren los despliegues más rápidos en lugar de los lentos.

En este proyecto se plantea aplicar la práctica llamada continuous delivery (CD), que trata de la implementación de un continuous delivery pipeline, de acuerdo con Shahina, Babara, y Zhub (2015), CD tiene por objetivo garantizar que la aplicación esté siempre en estado preparado para los despliegues a producción después de que la aplicación haya pasado las pruebas automatizadas. CD emplea un conjunto de prácticas, por ejemplo, Continuous Integration (CI), y despliegues automatizados para entregar software a un ambiente por ejemplo test, uat o producción. Esta práctica ofrece varios beneficios, como un menor riesgo al realizar despliegues, reducir costos y obtener comentarios del cliente más rápidamente. Savor, et al. (2017), manifiestan que las herramientas que conforman un continuous delivery pipeline son importantes, estos incrementan la productividad de los desarrolladores y decrementan el riesgo de fallos, porque ellas reducen el número operaciones manuales, en donde los errores son más probables de ocurrir.

Farley y Humble (2010), menciona que los despliegues son diferentes para cada lenguaje de programación, es decir un mismo procedimiento de despliegue no es el mismo para proyectos de diferentes lenguajes de programación; lo que conlleva a tener

muchos manuales de despliegues, los cuales no siempre están actualizados, dificultando así los despliegues. Además, el proceso de despliegue de requerimientos a los ambientes de test, uat y producción del proyecto portal de empleo son totalmente manuales, provocando que el proceso tome un tiempo promedio de 30 minutos, causando así que la aplicación no funcione durante todo ese proceso. Por todo lo mencionado anteriormente se plantea la implementación de un continuous delivery pipeline para medir su influencia en el proceso de despliegue de requerimientos del proyecto portal de empleo.

Los antecedentes considerados en la investigación son:

Gmeiner y Haslinger (2015), en su investigación donde plantean verificar la importancia de pruebas automatizadas en un continuous delivery pipeline de una compañía austriaca de negocios en línea, la cual ofrece apuestas deportivas, juegos en línea y servicios de entretenimiento. En esta investigación se menciona que las empresas que operan un negocio en línea necesitan realizar despliegues con frecuencia con nuevas características y correcciones de errores a ambientes de producción. Como conclusión se señala que un continuous delivery pipeline soporta las necesidades de negocio del mundo en línea, sin embargo, si no se implementa la automatización de tests se aumentará el riesgo de introducir problemas de calidad y efectos secundarios no deseados. Esta investigación aporta en este estudio, ya que recalca la importancia de tests automatizados, la cual puede ser análisis de calidad de código.

Chen (2016), en su investigación cuyo objetivo fue enfatizar los obstáculos en la adopción de continuous delivery en Paddy Power, la cual es una compañía que ofrece servicios de mercados regulados, a través de tiendas de apuestas, teléfonos e internet.

Los autores afirman que continuous delivery puede traer grandes beneficios, pero la implementación de un continuous delivery pipeline es un reto en algunas empresas. Como conclusión se señala que los principales obstáculos que se encontraron abarcan diversos ámbitos, incluidos los aspectos organizativos, culturales y de procesos. Aspecto organizativo fue cuando se necesitó acceso root de un servidor y otro equipo controlaba estos permisos, para llegar a una solución implicó mucha consulta y negociación durante seis meses. Aspecto cultural, ya que se tuvo que re-estructurar la organización para romper las barreras entre los equipos de desarrollo e infraestructura. Aspecto de procesos, cada vez que se quería hacer un despliegue a producción, esto debía ser aprobado, lo cual tomaba alrededor de 4 días. Esta investigación aporta en este estudio, ya que se menciona que se pueden presentar obstáculos en la implementación de un continuous delivery pipeline que una vez resueltas causarán grandes beneficios.

Cobanoglu (2016), en su investigación en la que adoptan la práctica continuous delivery en una aplicación llamada TAG (Tek Araba Gidelim), la cual fue desarrollada por un startup llamada Lojika. Los autores afirman que para aplicar continuous delivery, primero se tiene que adoptar continuous integration la cual tiene como objetivo que el software esté funcionando todo el tiempo considerando tests automatizados. Como conclusión se señala que continuous delivery es una disciplina ágil, la cual se puede implementar con éxito y así mejorar los procesos de desarrollo de software, también se concluye que la práctica de continuous delivery puede ser aplicada no sólo en grandes compañías, sino también en startups. Esta investigación aporta en este estudio, ya que se menciona que para adoptar la práctica continuous delivey se tiene que adoptar la práctica continuous integration, ya que sin esta no se tendrá un release a desplegar.

Wider y Deger (2017), en su investigación en donde realizan actualizaciones de precios de autos en un software de AutoScout24, la cual es una plataforma online especializada en la venta de vehículos de segunda mano. En esta publicación se afirma que la compañía trabajaba con una infraestructura muy antigua que no permitía que su aplicativo web funcione correctamente e impedía realizar despliegues rápidos y confiables. Para esto la compañía Thoughtworks diseñó un continuous delivery pipeline con herramientas de Amazon Web Services (AWS) y diferentes frameworks. Como conclusión se señala que la compañía AutoScout24 obtuvo un aplicativo web con buen rendimiento, responsivo, con soporte a una gran cantidad de usuarios a la vez y con un tiempo de despliegue que tomaba milisegundos gracias a la adopción de continuous delivery. Esta investigación aporta en este estudio, ya que se menciona que la tecnología Amazon Web Services permite implementar un continuous delivery pipeline, y con esta se puede reducir tiempos de despliegue.

Parizo (2018), en su investigación cuyo propósito fue buscar la mejor forma de resolución de problemas de infraestructura en CenturyLink, la cual es una compañía estadounidense de telecomunicaciones. Se menciona que siempre que existen problemas no se logra saber con exactitud cuáles fueron los pasos que se siguieron para llegar a un error por ejemplo en un despliegue. Como conclusión se menciona con respecto a la implementación del continuous delivery pipeline que, si algo va mal entonces el beneficio es que no se debe preguntar a nadie sobre qué fue lo que hicieron, en lugar de eso, ya que todo este proceso está en scripts, el problema se puede depurar. Sus resultados están basados en herramientas como TeamCity y Github. Esta investigación aporta en este estudio, ya que en caso se presenten errores en un despliegue, estos se podrán replicar porque se podrá saber los pasos exactos que se

siguieron para realizar el despliegue y de esa forma poder partir con la resolución del problema.

Zhu, et al. (2015), en su investigación en la que realizan la implementación de un continuous delivery pipeline en Etsy, la cual es un mercado en línea que se especializa en artículos hechos a mano, antigüedades y materiales para manualidades. Como conclusión de la implementación del continuous delivery pipeline se señala que los releases más complejos pasaron a realizarse de meses a días o incluso horas, se tuvo 4004 despliegues en el ambiente de producción en solo 6 meses, con un promedio de 20 releases al día y 10 commits por release. Sus resultados están basados en Amazon Web Services y la herramienta de configuración Chef. Esta investigación aporta en este estudio, ya que se recalca que el tiempo de despliegue, disminuye con un continuous delivery pipeline compuesto por AWS EC2 y AWS OpsWorks.

Kim (2014), en su investigación cuya meta fue disminuir el tiempo de desarrollo y despliegue de la compañía Hewlett-Packard (HP), la cual es de las mayores empresas de tecnologías de la información del mundo. En esta publicación se afirma que HP tenía problemas con el tiempo de desarrollo de nuevas características de firmwares de la familia de productos HP LaserJet, realizando 2 despliegues al año. Para esto los ingenieros de HP apostaron por adoptar continuous delivery. Como conclusión se señala que los despliegues pasaron a realizarse en solo 24 horas, mejorando la productividad y disminuyendo costos de producción. Esta investigación aporta en este estudio, ya que se denota que un continuous delivery también puede funcionar con firmwares y no solo aplicaciones web.

McFadden (2017), en su investigación en la que adoptan cloud computing y mejoran el rendimiento, escalabilidad y usabilidad en la compañía SparkPost, la cual es el proveedor de infraestructura de correo electrónico número 1 del mundo. En esta

publicación se afirma que en el año 2015 el rendimiento de la aplicación era muy bajo. Para esto adoptaron continuous delivery con tecnología de Amazon Web Services. Como conclusión se señala que SparkPost mejoró notablemente su rendimiento y pasó a realizar despliegues varias veces a la semana, también se redujeron los ciclos de desarrollo de software de 22 días a 10 días. Esta investigación aporta en este estudio, ya que se denota que un continuous delivery ayuda a disminuir tiempos de despliegues reduciendo así el tiempo de ciclo de desarrollo de software.

Stelligent (2017), en su investigación cuya meta fue desplegar nuevas características de su software con tan solo hacer clic en un botón en la compañía Sony Pictures, la cual es una distribuidora y productora estadounidense de cine, en esta publicación se afirma que hacer despliegues de su software tomaba meses por lo que implementaron un continuous delivery pipeline con tecnología Amazon Web Services. Como conclusión el continuous delivery pipeline les permite hacer despliegues automatizados en minutos y escalar rápidamente la infraestructura de sus servidores. Esta investigación aporta en este estudio, ya que se menciona que un continuous delivery pipeline ayuda a los desarrolladores a estar más enfocados en agregar nuevas características al software en lugar de preocuparse por los despliegues.

Stelligent (2017), en su investigación cuyo propósito fue realizar despliegues rápidos en lugar de esperar días o semanas hasta que se siga un proceso manual de despliegue en la compañía Advent, la cual es una compañía que fabrica software diseñado para automatizar la contabilidad de carteras para firmas de administración de inversiones, en esta publicación se afirma que un proceso de despliegue manual es lento y alarga el ciclo de desarrollo de software. Como conclusión se implementó un continuous delivery pipeline con tecnología Amazon Web Services logrando así realizar despliegues con tan solo hacer clic en un botón. Esta investigación aporta en este

estudio, ya que se menciona que con el uso de continuous delivery pipeline lograron mejorar la satisfacción de las necesidades de sus clientes.

Son pocos los resultados cuantitativos de los antecedentes por lo que solo se manejan resultados genéricos los cuales serán contrastados en la discusión.

La implementación de un continuous delivery pipeline permite llevar a cabo la práctica ágil denominada continuous delivery, cuyo objetivo es asegurarse que una aplicación siempre esté lista para que se despliegue a ambientes de producción, luego de que la aplicación haya pasado tests automatizados. Continuous delivery emplea un conjunto de prácticas como son: continuous integration y despliegues automatizados a diferentes ambientes ya sean test, uat o producción. Esta práctica ofrece varios beneficios como reducir el riesgo al realizar despliegues, menor costo y obtener retroalimentación del cliente rápidamente (Shahina, Babara, & Zhub, 2015).

Según Farley & Humble (2010), se consideran beneficios al implementar un continuous delivery pipeline, los cuales son:

El primer beneficio es que crea un proceso de release que es confiable y previsible, a su vez genera grandes reducciones en el tiempo de despliegue, y por lo tanto se obtienen características y retroalimentación de errores reportados por el cliente rápidamente.

El segundo beneficio es mayor alcance de encargados para realizar despliegues, lo cual significa que es un sistema pull, es decir este permite a encargados de operaciones y testers obtener la versión de la aplicación que deseen, en el ambiente que quieran. Usualmente obtener un buen build requiere de muchos emails enviados, con la

implementación de un pipeline de despliegue este problema es eliminado completamente, todos deben tener la habilidad de ver cuáles son los builds que están disponibles para ser desplegados en cualquier ambiente y hacer el despliegue con tan solo hacer clic en un botón. La facilidad de hacer un despliegue de cualquier versión del software en cualquier ambiente tiene muchas ventajas:

- Testers pueden seleccionar versiones antiguas de una aplicación para verificar cambios de comportamiento en nuevas versiones.
- Personal de soporte puede desplegar una versión de release de la aplicación en un ambiente para reproducir un defecto.
- Personal de operaciones o desarrolladores de software pueden seleccionar el mejor build de la lista de builds.

El tercer beneficio es la reducción de errores, lo cual significa, por ejemplo, desplegar la versión correcta de código con las urls correctas para un web service.

El cuarto beneficio es la reducción de estrés en los despliegues a ambientes de producción, lo cual significa que las personas que alguna vez han participado en un proyecto de desarrollo de software que se acerca a su fecha de lanzamiento son conscientes que estos son eventos realmente estresantes. El problema aquí es que los despliegues a producción son grandes eventos, lo cual provoca nerviosismo ya que, si algo se ejecuta de la manera incorrecta, provocará fallos en el servidor de producción. La clave para reducir el estrés es tener el proceso de despliegue automatizado, de esta manera los despliegues son rápidos y tienen poca probabilidad de que fallen, en caso el despliegue falle se puede volver a versiones anteriores en segundos.

Según Farley & Humble (2010), continuous delivery tiene cuatro principios:

El primer principio es crear un proceso repetible y confiable para hacer un release de un software, lo cual significa que realizar el despliegue de un software debe ser fácil, tan simple como hacer clic en un botón.

El despliegue de una aplicación en un servidor implica:

- Provisionar y administrar el entorno en el que la aplicación va a ejecutarse (configuración de hardware, software y servicios externos).
- Instalar la versión correcta de la aplicación.
- Configuración de la aplicación.

El segundo principio es automatizar casi todo, lo cual se refiere a que existen algunas cosas que son imposibles de automatizar, como son las pruebas exploratorias las cuales son realizadas por testers experimentados, se debe considerar automatizar lo más que se pueda, ya que se considera que la automatización es un prerrequisito para un pipeline de despliegue.

El tercer principio es mantener todo en un control de versiones, lo cual significa que todo lo que se necesita para generar un build y el release de una aplicación debe mantenerse en algún tipo de almacenamiento versionado.

El cuarto principio es calidad**,** lo cual se refiere a que defectos encontrados en etapas tempranas son más fáciles de corregir, el objetivo es encontrar defectos tan rápido como sea posible, el siguiente paso es corregirlo.

Otra terminología utilizada en la presente investigación es Continuous integration o integración continua, requiere que cada vez que un desarrollador de software haga cualquier cambio en el código, se debe generar un build de la aplicación entera y validarla con tests automatizados. De tal manera, si el proceso falla, el equipo de

desarrollo deja de hacer lo que está haciendo y corrigen el problema inmediatamente. El objetivo de continuous integration es que el software esté funcionando todo el tiempo considerando tests automatizados. Para adoptar continuous delivery se tiene que aplicar continuous integration, ya que continuous delivery necesita de un release, para que se realice el despliegue, trayendo como beneficio que hayan menos errores en la aplicación a desplegar, en caso se encuentren errores, estos se solucionarán rápidamente y serán menos costosas, ya que se da en etapas tempranas (Farley & Humble, 2010).

Según Farley & Humble (2010), la implementación de continuous integration involucra conocer algunos conceptos como:

- Control de versiones se refiere a que todo en un proyecto de software debe estar en un único repositorio de control de versiones: código fuente, scripts de base de datos, todo lo que se necesita para crear, instalar y ejecutar tu aplicación.
- Acuerdo del equipo se refiere a que continuous integration es una práctica no una herramienta, para lograrla exitosamente se requiere del compromiso y disciplina del equipo de desarrollo. Se requiere que todos los miembros del equipo de desarrollo hagan frecuentes cambios de acuerdo a la prioridad de tareas en el proyecto y resolver problemas que dejan sin funcionamiento a la aplicación.
- Creación de builds se trata de generar un build a partir del código fuente con la ejecución de una serie de comandos, cabe resaltar que el build será exitoso si pasa todos los tests automatizados.

Para que se lleve a cabo continuous integration correctamente se debe seguir las siguientes prácticas antes de empezar:

La primera práctica, se deben subir cambios de código fuente regularmente al repositorio, esto hace que los cambios sean pequeños y así sea difícil que falle un build, además tenemos recientes versiones del software para revertir cuando algo anda mal y también asegura que los cambios que abarcan varias hojas de código sean menos propensos a causar conflictos con trabajos de otros desarrolladores del mismo equipo (Farley & Humble, 2010).

La segunda práctica es crear un conjunto de tests automatizados, los cuales son opcionales pero esenciales para garantizar que la aplicación esté trabajando correctamente, pueden ser:

- Tests unitarios, están escritos para testear el comportamiento de pequeñas partes de la aplicación independientemente, no se requiere que la aplicación esté ejecutándose completamente para realizar los tests y se ejecutan rápidamente.
- Tests de calidad de código, para verificar la calidad de código, esto se puede realizar con herramientas como SonarQube, proporcionando la funcionalidad mejorar la calidad del código fuente (Chen L. , 2017).

Otra terminología utilizada en la presente investigación es despliegue, el despliegue de un software es el penúltimo paso en el ciclo de desarrollo de software, después de realizado el despliegue la organización cliente se verá beneficiada por usar el software instalado o actualizado, cualquier problema detectado después del despliegue será resuelto en la fase de mantenimiento (Subramanian, 2017).

Las herramientas y procesos para desplegar software a ambientes de producción son similares a migrar cambios entre ambientes de desarrollo, sin embargo cuando se despliega a ambientes de producción hay diferencias importantes y varios pasos

adicionales, porque el proceso de despliegue a producción depende de las políticas del departamento de TI, no hay un proceso prescrito para desplegar a ambientes de producción; sin embargo existen buenas prácticas para realizar despliegues exitosos (salesforce, 2017).

Es importante realizar despliegues durante el periodo en el cual los usuarios no están usando el software, además es aconsejable crear ambientes de test que permitan hacer pruebas de despliegue antes de hacer despliegues a producción, cabe resaltar que un ambiente de test es una copia exacta de un ambiente de producción (salesforce, 2017).

Tener en cuenta que todas las organizaciones de tecnologías de información (TI) son diferentes, los siguientes pasos de despliegue a producción son a nivel general y opcionales dependiendo de la organización.

1. Se anuncia una ventana de mantenimiento en la aplicación.
2. Se detiene la ejecución de la aplicación en el ambiente de producción.
3. Se crea un ambiente de test.
4. Se migran los cambios al ambiente de test creado anteriormente.
5. Se realiza el despliegue a producción.
6. Se deshabilita el anuncio de mantenimiento de la aplicación.

Según Mäntylä & Vanhanen (2010), las actividades de despliegue son:

- Comunicación con Stakeholder, mantener informados a los stakeholder acerca de los contenidos a ser desplegados en los ambientes de uat o producción; esto incluye a los clientes, estos deciden si autorizan o no la actualización del software.

- Preparación de la instalación, involucra integrar el producto software y programar la fecha de despliegue:

  Integrar todos los cambios de los desarrolladores en una sola versión del producto software.

  La programación de la fecha de despliegue para realizar la instalación o actualización se debe dar en cooperación con el cliente, programar una instalación es más flexible, ya que todavía no hay usuarios usando la aplicación a desplegar.

- Instalación:

  Pre-install checks de componentes externos de software como recursos (ram, disco duro...).

  Instalando el producto, la forma ideal es realizar un despliegue con un continuous delivery pipeline.

  Manteniendo información sobre productos desplegados, las compañías que desarrollan software deben tener documentación de despliegue de productos considerando las versiones, configuraciones de productos, información de lo que se hizo en el sitio web del cliente y cuáles fueron los errores en el momento de realizar el despliegue.

- Testear el producto instalado, a pesar de que el release es testeado durante el proceso de desarrollo, se debería probar las funcionalidades principales en producción una vez realizado el despliegue.

Otro concepto a considerar en esta investigación es la flexibilidad de despliegues, la cual se refiere a que desplegar una aplicación consta solamente de ejecutar despliegues automatizados escogiendo el ambiente y la versión de software (Farley & Humble, 2010).

Según Raphael (2018), los ambientes pueden ser:

- Desarrollo, el servidor de desarrollo se utiliza para probar el código directamente por los desarrolladores de software.
- Test, una vez que se haya completado el trabajo los desarrolladores desplegarán el software finalizado en este ambiente, en donde el tester ejecutará casos de uso para garantizar que el producto esté funcionando como debería, si el tester encuentra errores u otros problemas creará tareas para que los desarrolladores de software lo examinen y solucionen.
- Pruebas de aceptación del usuario (UAT), es un ambiente en donde el cliente accede y ve cómo funcionará el software en producción, aquí podrá verificar que los requerimientos que solicitó funcionen correctamente.
- Producción, este ambiente es la ubicación final de los trabajos terminados y aprobados. Esta etapa el trabajo se considera completo y aprobado.

Influencia positiva en el proceso de despliegues se refiere a:

- Disminuir o mantener los valores de los indicadores de calidad que se presentan en la operacionalización de variables (ver Anexo 1) los cuales son: número de errores, número de vulnerabilidades, número de code smells, tiempo de deuda técnica, porcentaje de código duplicado y bloques duplicados (Cecil, 2009).

Cabe mencionar que el porcentaje de código duplicado y bloques duplicados pueden aumentar en el transcurso en el que se resuelven los atributos con mayor prioridad, todo por el hecho de que el código duplicado es de baja prioridad para solucionarse, pero esto va a ser temporal, ya que una vez resueltos los indicadores con mayor prioridad se pasará a resolver los indicadores de prioridad baja como es el del código duplicado y se conseguirán disminuir los valores de estos indicadores.

Para que el software se considere de calidad alta todos los indicadores de calidad deben ser cero o se deben resolver todos los indicadores de alta prioridad, caso contrario el software se considera de calidad baja (Cecil, 2009).

- Disminuir los valores de los indicadores de automatización de despliegues que se presentan en la operacionalización de variables (ver Anexo 1) los cuales son: tiempo de solución de requerimientos, tiempo que toma integrar todo el código fuente, número de errores al realizar despliegue y tiempo de despliegue (Farley & Humble, 2010).

Un requerimiento describe lo que se hará en el software, en esta investigación se consideran dos tipos:

- Fallo: El sistema no funciona como debería (mantenimiento correctivo).
- Cambio: Agregar una nueva funcionalidad al sistema (mantenimiento perfectivo).

Otra terminología utilizada en la presente investigación es calidad de software, según Xiaoqing (2009), la calidad de software facilita la realización de despliegues, además

la calidad del proceso de desarrollo de software abarca tiempo de entrega de software e integración.

Según Gaudin (2016), SonarQube por defecto, soporta los siguientes lenguajes: Java, C/C++, Php, JavaScript, C#; Además, tal y como indica la página oficial de sonarsource (2018), SonarQube muestra los siguientes indicadores en base a reglas que define el estándar CWE:

- Error, es un problema que representa algo malo en el código, esto va a desenlazar problemas en el futuro.
- Vulnerabilidad, es un problema relacionado a la seguridad que representa una entrada para los atacantes.
- Code smell, se refiere a un problema relacionado al mantenimiento del código, significa que, si mayor es el code smell, entonces va a ser más difícil hacerle mantenimiento al software y posiblemente se agregarán más errores.
- Deuda técnica, es el tiempo estimado requerido para solucionar todos los problemas de mantenimiento es decir code smell.
- Porcentaje de código duplicado, es el porcentaje de líneas de código que se repiten en todo el proyecto software.
- Bloque duplicado, conjunto de líneas de código que se repiten en todo el proyecto software.

El indicador de porcentaje duplicado y bloque duplicado cuenta como severidad menor es decir no tiene mucha prioridad, todos los indicadores a excepción de porcentaje de código duplicado y bloques duplicados tienen un atributo llamado severity, la cual puede ser: Blocker, Critical, Major, Minor o Info.

Según Campbell (2017), el orden en la que se deben resolver los indicadores es en base a las prioridades, es decir primero Blocker, segundo Critical, tercero Major y cuarto Minor.

- Blocker, es un bug con alta probabilidad de impactar el comportamiento de la aplicación, por ejemplo: pérdida de memoria.
- Critical, ya sea un error con baja probabilidad de impactar el comportamiento de la aplicación o un problema que representa una falla de seguridad.
- Major, es un fallo de calidad que puede impactar la productividad del desarrollador, por ejemplo: parámetros de una función no utilizados.
- Minor, Fallo de calidad que puede afectar ligeramente a la productividad del desarrollador, por ejemplo: las líneas no deben ser demasiado largas.
- Info, Ni un error ni un defecto de calidad, sólo un hallazgo no identificado.

El término Kanban proviene de Japón gracias al sistema de producción de Toyota, que es muy conocido en círculos reducidos, sus principios básicos: fabricación ajustada, desarrollo continuo, orientación al cliente, etc; tiene una traducción literal. "Kan" significa visible o visual y "ban" significa una tarjeta o tablero. Las tarjetas de la metodología Kanban se utilizan en todas las plantas de Toyota para mantener la gestión del inventario sin almacenes desordenados y talleres con suficiente acceso a las piezas (Laptick, 2016).

En kanban se consideran 3 principios mencionados por Laptick (2016):

- Visualiza producción:
    - Dividir el trabajo en tareas.
    - Escribir cada tarea en una tarjeta.

- Poner tarjetas en una pizarra (board).

- Limita el trabajo en progreso (WIP)
- Medir tiempo tomado en tareas.

En Kanban se visualiza el flujo de trabajo: el trabajo se divide en pequeños items discretos y se escribe en una tarjeta que está pegada a un tablero, el cual tiene columnas diferentes y a medida que el trabajo avanza por diferentes etapas (por listo, en progreso, listo para su revisión, etc.), la tarjeta se mueve.

En Kanban, el número de elementos que pueden estar en progreso en cualquier momento es estrictamente limitado.

Otras terminologías utilizadas en la presente investigación son:

Candidato release, es un build, producto de cada commit hecho por un desarrollador de software que no se sabe si va pasar todos los tests automatizados exitosamente (Farley & Humble, 2010).

Python, es un lenguaje de programación que permite trabajar más rápido e integrar sus sistemas de manera más eficaz, se puede aprender a usarlo y ver ganancias casi inmediatas (Python, 2017).

Tester, su objetivo es encontrar fallas o defectos dentro del programa que está probando (Holland, 2012).

Startup, es una empresa en etapa temprana que se basa en un negocio que será escalable rápida y fácilmente, haciendo uso de tecnologías digitales (Morelos, 2018).

Un build es una versión de un programa y se identifica mediante un número, es una parte importante del proceso de desarrollo, para garantizar un producto final confiable (Rouse, 2007).

Un release es la versión final de una aplicación, puede ir precedido por diferentes builds de prueba (Rouse, 2008).

El ciclo de desarrollo de software (SDLC) es una terminología utilizada para explicar cómo se entrega software a un cliente en una serie de fases. Estas fases llevan el software desde la ideación hasta la entrega (Swersky, 2018).

1. Planeamiento: Incluye plan de proyecto y estimaciones de costo.
2. Requerimiento: El cliente debe comunicarse con los equipos de TI para transmitir sus requisitos para nuevos desarrollos y mejoras.
3. Diseño y prototipado: Una vez que se entienden los requisitos, los arquitectos y desarrolladores de software pueden comenzar a diseñar el software.
4. Desarrollo de software: Esta fase produce el software en desarrollo.
5. Testing: Tests unitarios, tests de calidad; lo ideal es que sean automatizadas para garantizar calidad.
6. Despliegue: Proceso en el cual se entrega el software funcionando en los servidores del cliente.
7. Mantenimiento: Solución de errores del software reportados en el ambiente de producción, en este paso se consideran los anteriores necesarios para no introducir errores en la aplicación.

Commit es un cambio de uno o un conjunto de archivos, permite mantener grabado los cambios, cuándo y quién realizó cambios en dichos archivos, usualmente también contiene un mensaje con una breve descripción (GitHub, 2018).

Amazon Web Services (AWS) es una plataforma segura de servicios en la nube que ofrece potencia de cómputo, almacenamiento de bases de datos, entrega de contenido y otras funcionalidades (aws, 2018).

Bitbucket es una solución de gestión de repositorio git diseñada para equipos profesionales que proporciona un lugar central para administrar los repositorios de código fuente (Atlassian, 2018).

Bamboo es una herramienta de integración y entrega continua que vincula las compilaciones, pruebas y lanzamientos automatizados en un solo flujo de trabajo (Atlassian, 2018).

SonarQube es una herramienta de revisión automática de código que puede integrarse con un flujo de trabajo existente para habilitar la inspección continua de código (SonarQube, 2018).

Despliegue de requerimiento se refiere a pasar cambios de código de un ambiente a otro, por ejemplo, pasar cambios de código que conforman una nueva funcionalidad en el portal de empleo del ambiente de desarrollo al ambiente de producción (Mäntylä & Vanhanen, 2010).

1.2. Formulación del problema

¿Cómo influye la implementación de un continuous delivery pipeline en el proceso de despliegue de requerimientos de un software?

1.3. Limitaciones

- Carencia de antecedentes locales, en relación a la implementación del continuous delivery pipeline.
- Pruebas de la implementación realizadas con una simulación del ambiente de producción con 20 requerimientos debido a que no se puede hacer uso del nombre de la institución para la que se desarrolló el proyecto, lo cual limita ver el comportamiento en un entorno real.

- Documentación a nivel general del proyecto portal de empleo, ya que es información confidencial.

1.4. Objetivos

1.4.1. Objetivo general

Determinar la influencia de un continuous delivery pipeline en el proceso de despliegue de requerimientos de un software.

1.4.2. Objetivos específicos

- Implementar el continuous delivery pipeline para probar un software basado en Python con la metodología Kanban.
- Determinar la influencia de la implementación del continuous delivery pipeline en la calidad de código de un software basado en Python.
- Determinar la influencia de la implementación del continuous delivery pipeline en la automatización de despliegue de un software basado en Python.

1.5. Hipótesis

1.5.1. Hipótesis general

La implementación de un continuous delivery pipeline influye positivamente en el proceso de despliegue de requerimientos de un software.

1.5.2. Hipótesis específicas

- La implementación de un continuous delivery pipeline influye positivamente en el número de errores de un software basado en Python.
- La implementación de un continuous delivery pipeline influye positivamente en el número de vulnerabilidades de un software basado en Python.
- La implementación de un continuous delivery pipeline influye positivamente en el número de code smells de un software basado en Python.

- La implementación de un continuous delivery pipeline influye positivamente en el tiempo de deuda técnica de un software basado en Python.
- La implementación de un continuous delivery pipeline influye positivamente en el porcentaje de código duplicado de un software basado en Python.
- La implementación de un continuous delivery pipeline influye positivamente en el número de bloques duplicados de un software basado en Python.
- La implementación de un continuous delivery pipeline influye positivamente en el tiempo de solución de requerimientos de un software basado en Python.
- La implementación de un continuous delivery pipeline influye positivamente en el tiempo de integración de código de un software basado en Python.
- La implementación de un continuous delivery pipeline influye positivamente en el tiempo que toma realizar despliegues de un software basado en Python.
- La implementación de un continuous delivery pipeline influye positivamente en el número de errores al realizar despliegues de un software basado en Python.

CAPÍTULO II. METODOLOGÍA

2.1. Tipo de investigación

2.1.1. Según el propósito

Aplicada.

2.1.2. Según el diseño de investigación

Experimental, de tipo cuasi experimental.

Diseño de investigación es experimental, ya que se administran estímulos como es el pre y post test; de tipo cuasi experimental, ya que se realizan pruebas con requerimientos en una aplicación real (Sampieri, 2014).

2.2. Materiales, instrumentos y métodos

2.2.1. Población y Muestra

Para la presente investigación para que la muestra sea representativa se considera toda la población, la cual consiste de 20 requerimientos, los cuales fueron reportados en un contrato de mantenimiento del proyecto portal de empleo, la cual consistió en la solución de errores o agregar nuevas funcionalidades.

2.2.2. Métodos e Instrumentos

2.3. Técnicas e instrumentos de recolección de datos

2.3.1. Técnicas

Para la presente investigación se ha tomado como técnica la observación.

2.3.2. Instrumentos

- Como instrumento se considera la guía de observación, la cual se muestra en el Anexo 3, validada por nuestros expertos. Además, se consideró aplicar un pre test y post test.
- Para la confiabilidad del instrumento se realizó a través del Alpha de Cronbach en el software IBM SPSS Statistics versión 23, la cual se muestra en el Anexo 4.

Interpretación

El resultado del Alfa de Cronbach es de 0.719, este valor es mayor a 0.7 entonces el instrumento tiene una confiabilidad aceptable.

2.4. Procedimiento

Para lograr medir los resultados y las conclusiones del proyecto de investigación, el procesamiento de datos comienza cuando se aplica el pre-test, luego se ingresa esta información al software IBM SPSS Statistics versión 23 para realizar el análisis de Alfa de Cronbach, luego se aplica estadística descriptiva para saber máximos, mínimos y promedios; después se realiza el post-test, se ingresa esta información al software IBM SPSS Statistics versión 23, luego se aplica estadística descriptiva para saber máximos, mínimos y promedios; finalmente se aplica la prueba de Wilcoxon para muestras relacionadas y así contrastar la hipótesis (ver Figura 1).

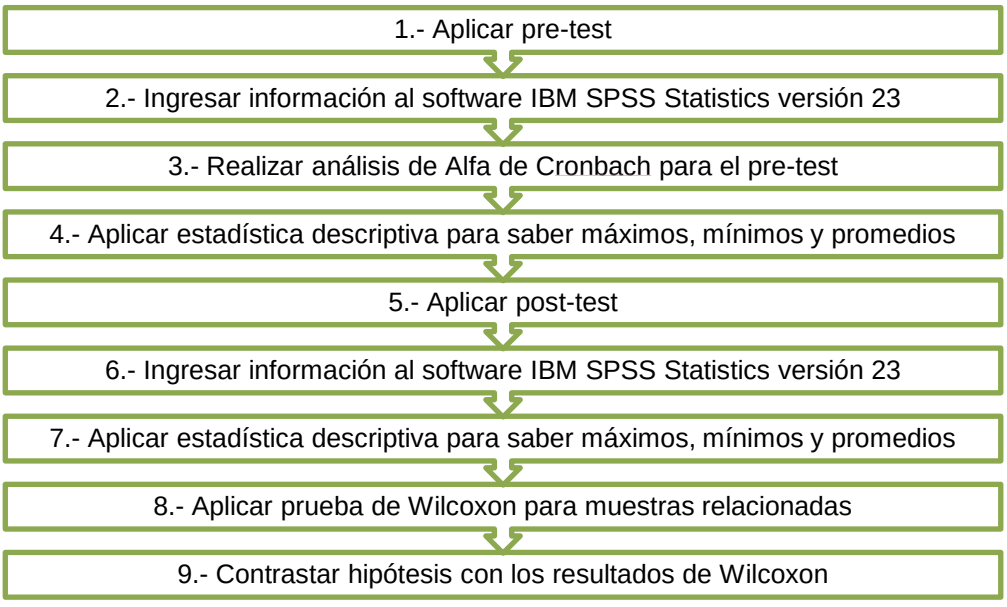


Figura 1. Procesamiento de datos

CAPÍTULO III. RESULTADOS

Objetivo Específico 1

Implementar el continuous delivery pipeline para probar un software basado en Python con la metodología Kanban.

El desarrollo se dio considerando tres estados: To do, In Progress y Done, se utilizó como herramienta de apoyo al software Jira, siguiendo los principios de la metodología kanban:

- **Visualiza producción:**
  - Se dividió todo el desarrollo de la investigación en tareas las cuales se muestran en la siguiente tabla (ver Tabla 1).

Tabla 1 *Tareas kanban del desarrollo de la investigación*

Tipo de tarea	Key	Status	Descripción	Tiempo en minutos	Total
Task	POR-32	Done	Redacción de resultados	4800	4800
Task	POR-31	Done	Análisis de los datos obtenidos	2400	2400
Task	POR-30	Done	Aplicación post-test		535
-> Sub-task	POR-86	Done	-> Requerimiento 20	20	
-> Sub-task	POR-85	Done	-> Requerimiento 19	25	
-> Sub-task	POR-74	Done	-> Requerimiento 18	17	
-> Sub-task	POR-73	Done	-> Requerimiento 17	20	
-> Sub-task	POR-72	Done	-> Requerimiento 16	19	
-> Sub-task	POR-71	Done	-> Requerimiento 15	19	
-> Sub-task	POR-70	Done	-> Requerimiento 14	16	
-> Sub-task	POR-69	Done	-> Requerimiento 13	20	
-> Sub-task	POR-68	Done	-> Requerimiento 12	35	
-> Sub-task	POR-54	Done	-> Requerimiento 11	24	
-> Sub-task	POR-53	Done	-> Requerimiento 10	26	
-> Sub-task	POR-52	Done	-> Requerimiento 9	28	
-> Sub-task	POR-51	Done	-> Requerimiento 8	28	
-> Sub-task	POR-50	Done	-> Requerimiento 7	30	
-> Sub-task	POR-49	Done	-> Requerimiento 6	31	
-> Sub-task	POR-48	Done	-> Requerimiento 5	32	
-> Sub-task	POR-47	Done	-> Requerimiento 4	33	
-> Sub-task	POR-46	Done	-> Requerimiento 3	34	
-> Sub-task	POR-45	Done	-> Requerimiento 2	37	
-> Sub-task	POR-44	Done	-> Requerimiento 1	41	

Task	POR-29	Done	Aplicación pre-test		545
-> Sub-task	POR-84	Done	-> Requerimiento 20	20	
-> Sub-task	POR-83	Done	-> Requerimiento 19	25	
-> Sub-task	POR-67	Done	-> Requerimiento 18	21	
-> Sub-task	POR-66	Done	-> Requerimiento 17	18	
-> Sub-task	POR-65	Done	-> Requerimiento 16	19	
-> Sub-task	POR-64	Done	-> Requerimiento 15	22	
-> Sub-task	POR-63	Done	-> Requerimiento 14	22	
-> Sub-task	POR-62	Done	-> Requerimiento 13	20	
-> Sub-task	POR-61	Done	-> Requerimiento 12	39	
-> Sub-task	POR-43	Done	-> Requerimiento 11	43	
-> Sub-task	POR-42	Done	-> Requerimiento 10	38	
-> Sub-task	POR-41	Done	-> Requerimiento 9	34	
-> Sub-task	POR-40	Done	-> Requerimiento 8	33	
-> Sub-task	POR-39	Done	-> Requerimiento 7	31	
-> Sub-task	POR-38	Done	-> Requerimiento 6	29	
-> Sub-task	POR-37	Done	-> Requerimiento 5	29	
-> Sub-task	POR-36	Done	-> Requerimiento 4	28	
-> Sub-task	POR-35	Done	-> Requerimiento 3	26	
-> Sub-task	POR-34	Done	-> Requerimiento 2	25	
-> Sub-task	POR-33	Done	-> Requerimiento 1	23	
Task	POR-28	Done	Caso de uso realizar rollback	240	240
Task	POR-27	Done	Caso de uso realizar despliegue	240	240
Task	POR-26	Done	Caso de uso subir cambios al repositorio	240	240
Task	POR-25	Done	Elaboración de manuales técnicos	1200	1200
Task	POR-24	Done	Implementación del pipeline		13200
-> Sub-task	POR-82	Done	-> Creación de plan de despliegue	2400	
-> Sub-task	POR-81	Done	-> Creación de plan de integración	3600	
-> Sub-task	POR-80	Done	-> Creación de instancia AWS para Ambiente de producción	1200	
-> Sub-task	POR-79	Done	-> Creación de instancia AWS para SonarQube	1200	
-> Sub-task	POR-78	Done	-> Creación de instancia AWS para Bamboo	1200	
-> Sub-task	POR-77	Done	-> Preparación de ambiente de producción	1200	
-> Sub-task	POR-76	Done	-> Instalación de SonarQube	1200	
-> Sub-task	POR-75	Done	-> Instalación de Bamboo	1200	
Task	POR-23	Done	Elegir Softwares a utilizar	3360	3360
Task	POR-20	Done	Definir indicadores	2400	2400
Task	POR-19	Done	Descripción de la metodología	2400	2400

Task	POR-18	Done	Bases teóricas	3600	3600
Task	POR-17	Done	Antecedentes	4800	4800
Task	POR-16	Done	Objetivos	480	480
Task	POR-15	Done	Limitaciones	480	480
Task	POR-14	Done	Formulación del problema	1200	1200
Task	POR-13	Done	Realidad problemática	2400	2400
Total		44520		44520	

Fuente: Jira

- Se creó una tarjeta por cada tarea.
- Se creó una pizarra kanban en Jira con todas las tareas (ver Figura 2).

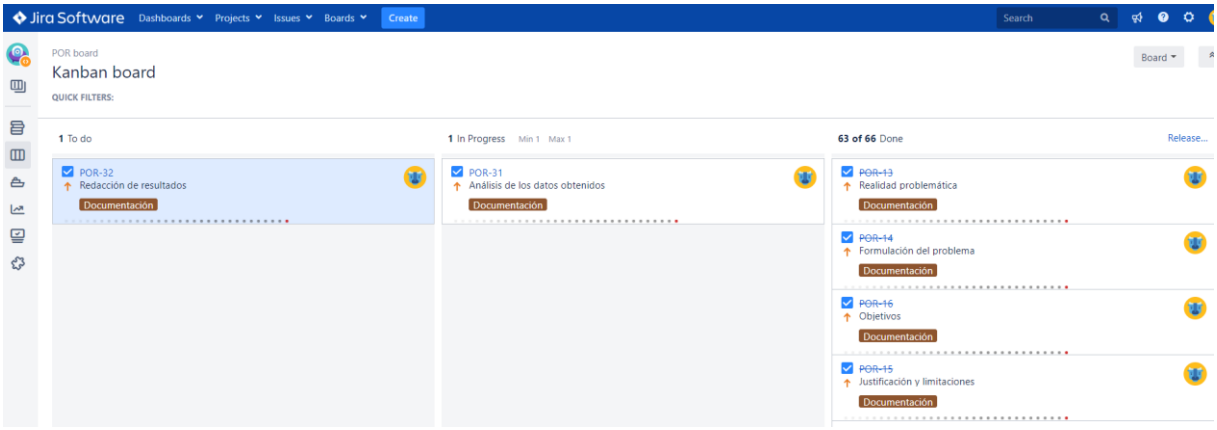


Figura 2. Tablero Kanban en Jira

Fuente: Jira

- **Limita el trabajo en progreso (WIP):** Se configuró un máximo de tareas que pueden estar en progreso, el cual es uno (ver Figura 3).

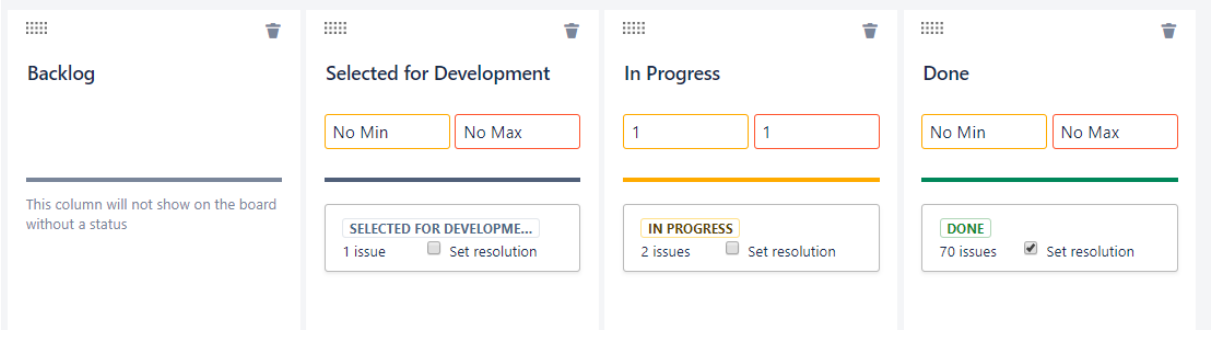


Figura 3. Configuración WIP en Jira

Fuente: Jira

- **Medir tiempo tomado en tareas:** Se registró el tiempo que tomó cada tarea (ver Figura 4).

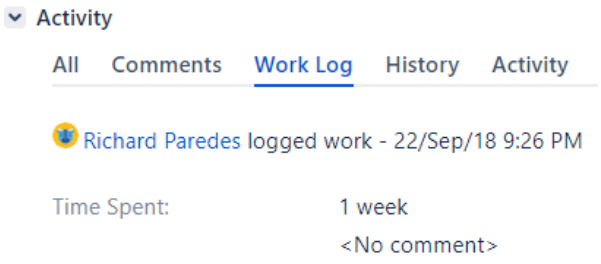


Figura 4. Registro de tiempo de tarea en Jira

Fuente: Jira

En la Tabla 2 se muestra el presupuesto inicial del proyecto divido en recursos humanos, materiales y servicios, mostrando un presupuesto total de 4 850 soles.

Tabla 2 *Presupuesto inicial del proyecto*

Recursos Humanos				
Descripción	U.med	Cantidad	Precio U.	Precio T.
Responsable del proyecto de tesis	unidad	1	S/ 300.00	S/ 300.00
Subtotal de Recursos				S/ 900.00

Materiales				
Descripción	U.med	Cantidad	Precio U.	Precio T.
Laptop	unidad	1	S/ 3,000.00	S/ 3,000.00
Bamboo	unidad	1	S/ 50.00	S/ 50.00
SonarQube	unidad	1	S/ -	S/ -
Bitbucket	unidad	1	S/ -	S/ -
Subtotal de Recursos				S/ 3,050.00

Servicios				
Descripción	U.med	Cantidad	Precio U.	Precio T.
Servidores AWS	mes	5	S/ 50.00	S/ 250.00
Impresiones	unidad	5	S/ 100.00	S/ 500.00
Acceso a internet	unidad	1	S/ 100.00	S/ 100.00
Luz	unidad	1	S/ 50.00	S/ 50.00
Subtotal de Recursos				S/ 900.00

Presupuesto Total	S/ 4,850.00

En la Figura 5 se muestra el diseño del continuous delivery pipeline, el cual integra diferentes herramientas, entre las cuales está un repositorio de código fuente (Bitbucket), herramienta de integración continua (Bamboo) y un analizador de calidad de código (SonarQube).

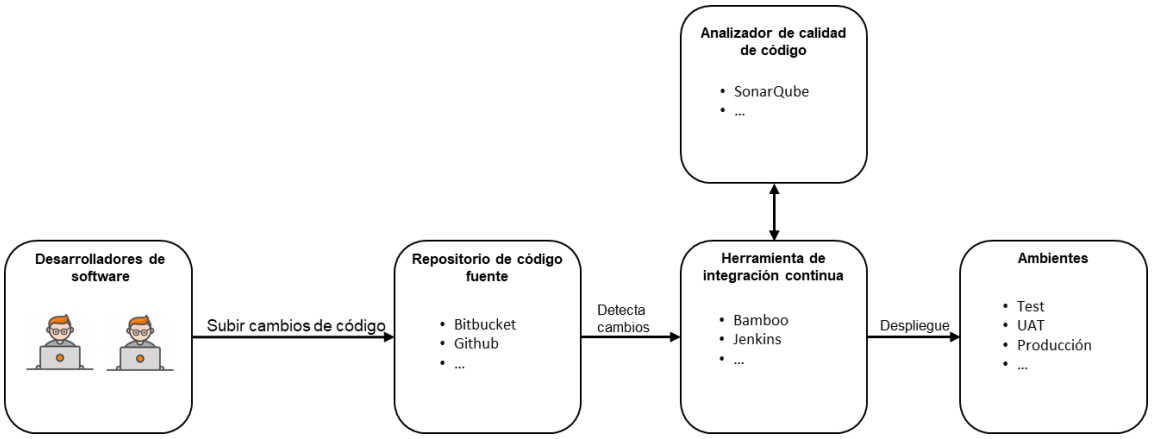


Figura 5. Diseño del continuous delivery pipeline

Los manuales técnicos de la implementación son los siguientes:

- **Bamboo**
  - **Instalación** (ver Anexo 8)
  - **Configuración análisis de calidad** (ver Anexo 10)
  - **Configuración despliegue automatizado** (ver Anexo 11)
- **SonarQube**
  - **Instalación** (ver Anexo 9)

El funcionamiento del continuous delivery pipeline comienza cuando uno o más desarrolladores de software hacen cambios en el proyecto portal de empleo, estos cambios los suben al repositorio de código fuente Bitbucket, luego estos cambios son detectados automáticamente por la herramienta de integración continua Bamboo, después Bamboo automáticamente hace que SonarQube realice un análisis de calidad de código, luego el

desarrollador con ayuda de Bamboo realiza un despliegue automatizado en la fecha dada por el cliente al ambiente de test, uat o producción.

Los casos de uso para el empleo del continuous delivery pipeline son los siguientes:

- **Subir cambios de código al repositorio** (ver Anexo 12)
- **Realizar despliegue** (ver Anexo 13)
- **Realizar rollback** (ver Anexo 14)

Sin el continuous delivery pipeline los requerimientos se resuelven siguiendo los siguientes pasos:

1. Se reporta un requerimiento.
2. Se reproduce el error en el ambiente de test.
3. Se resuelve el requerimiento en el ambiente de desarrollo.
4. Se realiza el despliegue al ambiente de test.
5. Se envía un correo al cliente diciendo que validen la solución del requerimiento.
6. El cliente acepta la solución.
7. Se realiza el despliegue al ambiente de producción.

Los despliegues se realizan manualmente en los siguientes ambientes: test, uat y producción.

Con el continuous delivery pipeline implementado estos pasos siguen siendo los mismos con la diferencia de que los despliegues son automatizados y va mejorando la calidad del software conforme se desarrollan requerimientos, además ayuda a mejorar el tiempo de ciclo de desarrollo de software en la fase de mantenimiento, la cual involucra las siguientes fases:

- Testing: con los tests de calidad realizados con SonarQube.
- Despliegue: con los despliegues automatizados realizados con Bamboo.

En las pruebas realizadas se usó únicamente un ambiente el cual simula ser test, uat y producción, ya que estos son copias idénticas, además no se consideran testers y tampoco un cliente aceptando requerimientos.

Objetivo Específico 2

Determinar la influencia de la implementación del continuous delivery pipeline en la calidad de código de un software basado en Python.

Los indicadores de calidad de software de la variable dependiente presentes en el instrumento se muestran en la Tabla 3, cada una denotada por letras, todos los indicadores fueron tomados de la operacionalización de variables (ver Anexo 1).

Tabla 3 *Indicadores de calidad de software*

	Indicadores
a	Número de errores
b	Número de vulnerabilidades
c	Número de code smells
d	Tiempo de deuda técnica en días
e	Porcentaje de código duplicado
f	Bloques duplicados

La recolección de información se dio por medio del instrumento que fue debidamente validado, la cual se muestra en el Anexo 3. El orden en el que se realizó la resolución de requerimientos fue la siguiente: REQ-001, REQ-002, REQ-003, REQ-004, REQ-005, REQ-006, REQ-007, REQ-008, REQ-009, REQ-010, REQ-011, REQ-012, REQ-013, REQ-014, REQ-015, REQ-016, REQ-017, REQ-018, REQ-019, REQ-020.

Por requerimiento se refiere a todo fallo (el sistema no funciona como debería), cambio (agregar una nueva funcionalidad al sistema) en el proyecto portal de empleo. Como ejemplo de un requerimiento se muestra en la Tabla 4.

Tabla 4 *Ejemplo de requerimiento tipo cambio*

Código	**REQ-001**
Título	Agregar dropdownlist en las reglas de acceso
Descripción	El dropdownlist debe permitir escoger las situaciones posibilitando así generar reglas con diferentes situaciones

PRE-TEST

El pre-test consistió en la resolución de 20 requerimientos, los cuales se muestran en el Anexo 6, la resolución de los requerimientos se dio de la manera en la que se trabaja actualmente (sin análisis de calidad de código y con despliegues manuales), es decir sin el continuous delivery pipeline. El procedimiento consistió en resolver el requerimiento y realizar el despliegue manualmente (ver Tabla 5).

Tabla 5 *Resultado pre-test de calidad de software*

	Indicadores					
Requerimiento	**a**	**b**	**c**	**d**	**e**	**f**
REQ-001	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-002	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-003	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-004	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-005	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-006	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-007	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-008	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-009	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-010	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-011	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-012	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-013	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-014	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-015	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-016	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-017	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-018	185,00	1,00	490,00	6,00	30,60	2000,00
REQ-019	173,00	1,00	486,00	6,00	30,60	2000,00
REQ-020	173,00	1,00	486,00	6,00	30,60	2000,00

Fuente: Instrumento guía de observación

Interpretación

- Se muestran valores elevados lo que indica que la calidad del proyecto es baja, para que el software se considere de calidad alta todos los indicadores de calidad deben ser cero o se deberían resolver todos los indicadores con mayor prioridad.

- Se muestra que los valores después de la resolución de cada requerimiento no aumentaron, lo cual indica que la calidad del software no bajó, en lugar de eso la calidad mejoró ligeramente en la solución de los dos últimos requerimientos.

POST-TEST

El post-test consistió en la resolución de 20 requerimientos, los cuales se muestran en el Anexo 6, la resolución de los requerimientos se dio con el uso del continuous delivery pipeline, lo que permitió realizar análisis de calidad de código y despliegues automatizados. El procedimiento consistió en resolver el requerimiento, luego se realizó un análisis de calidad de código, se mejoró la calidad de código con ayuda de SonarQube un tiempo promedio de 8.2 minutos y se realizó el despliegue automatizado (ver Tabla 6).

Tabla 6 *Resultado post-test de calidad de software*

	Indicadores					
Requerimiento	**a**	**b**	**c**	**d**	**e**	**f**
REQ-001	181,00	1,00	488,00	6,00	30,60	2000,00
REQ-002	177,00	1,00	486,00	6,00	30,60	2000,00
REQ-003	173,00	1,00	484,00	6,00	30,60	2000,00
REQ-004	169,00	1,00	482,00	6,00	30,60	2000,00
REQ-005	165,00	1,00	480,00	6,00	30,60	2000,00
REQ-006	161,00	1,00	477,00	6,00	30,60	2000,00
REQ-007	157,00	1,00	474,00	6,00	30,60	2000,00
REQ-008	153,00	1,00	470,00	6,00	30,70	2000,00
REQ-009	149,00	1,00	467,00	6,00	30,70	2000,00
REQ-010	145,00	1,00	464,00	6,00	30,70	2000,00
REQ-011	141,00	1,00	461,00	6,00	30,70	2000,00
REQ-012	138,00	1,00	459,00	6,00	24,50	1300,00
REQ-013	136,00	0,00	452,00	6,00	21,10	831,00
REQ-014	133,00	0,00	449,00	6,00	17,80	466,00
REQ-015	130,00	0,00	447,00	6,00	15,20	219,00
REQ-016	111,00	0,00	441,00	6,00	13,60	163,00
REQ-017	87,00	0,00	432,00	6,00	12,80	133,00
REQ-018	66,00	0,00	408,00	5,00	12,70	109,00
REQ-019	52,00	0,00	408,00	5,00	12,50	108,00
REQ-020	39,00	0,00	397,00	5,00	12,10	99,00

Fuente: Instrumento guía de observación

Interpretación

- **Indicador a - Número de errores**

  Se puede apreciar que el número de errores, ha ido disminuyendo conforme se resolvían los requerimientos, lo cual significa que la calidad de código del software mejoró.

- **Indicador b - Número de vulnerabilidades**

  Se puede apreciar que el número de vulnerabilidades paso a ser cero, lo cual significa que la calidad de código del software mejoró.

- **Indicador c - Número de code smells**

  Se puede apreciar que el número de code smells ha ido disminuyendo conforme se resolvían los requerimientos, lo que significa que la calidad de código del software mejoró.

- **Indicador d - Tiempo de deuda técnica en días**

  Se puede apreciar que el tiempo de deuda técnica disminuyó un día, lo cual significa que la calidad de código del software mejoró.

- **Indicador e - Porcentaje de código duplicado**

  Se puede apreciar que el porcentaje de código duplicado disminuyó, lo cual significa que la calidad de código del software mejoró.

- **Indicador f - Bloques duplicados**

  Se puede apreciar que el número de bloques duplicados disminuyó, lo cual significa que la calidad de código del software mejoró.

Resultado

Comparando los resultados del pre-test y post-test mostrados en las siguientes figuras, se puede apreciar que el continuous delivery pipeline mejora la calidad de código del software, ya que todos los indicadores han disminuido sus valores con la solución de 20

requerimientos, por lo que podemos concluir que, si se resuelven más requerimientos, entonces se puede suponer que todos los valores llegarán a ser cero, significando que la calidad de software llegará a ser alta.

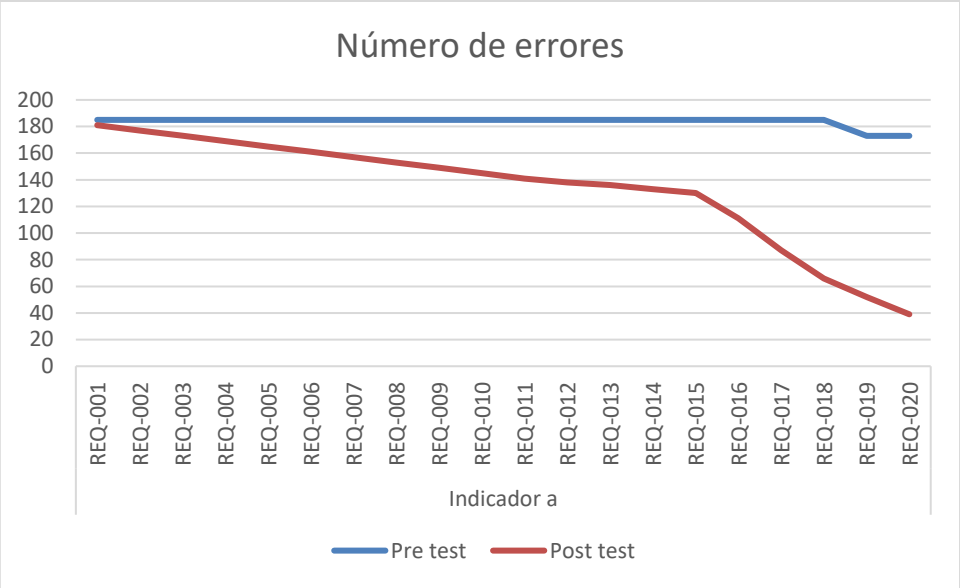


Figura 6. Comparación de número de errores de Pre test y Post test

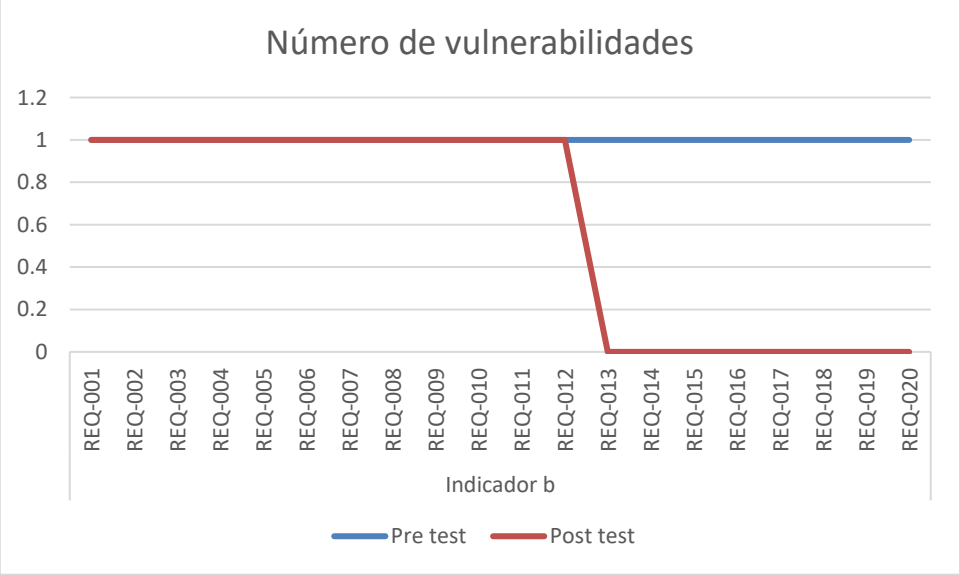


Figura 7. Comparación de número de vulnerabilidades de Pre test y Post test

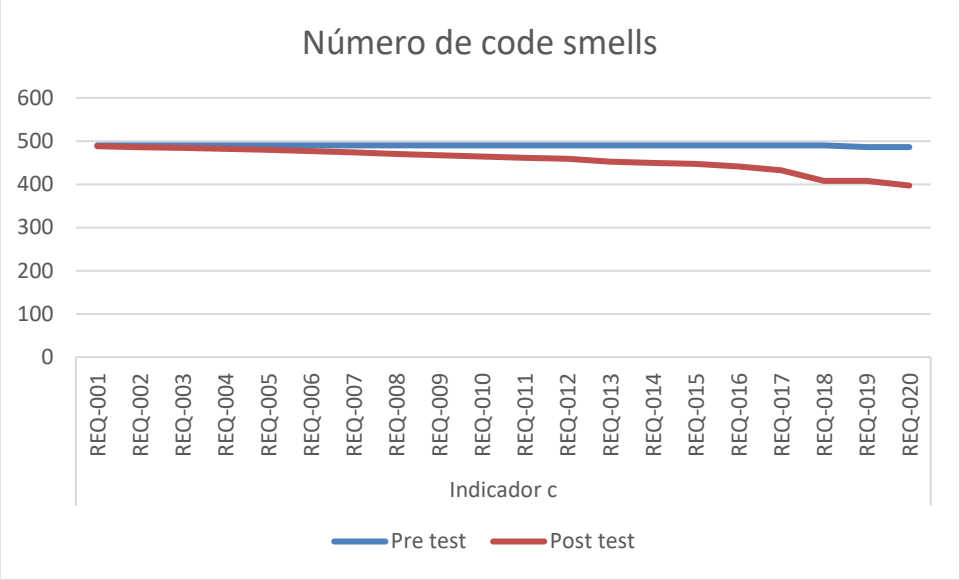


Figura 8. Comparación de número de code smells de Pre test y Post test

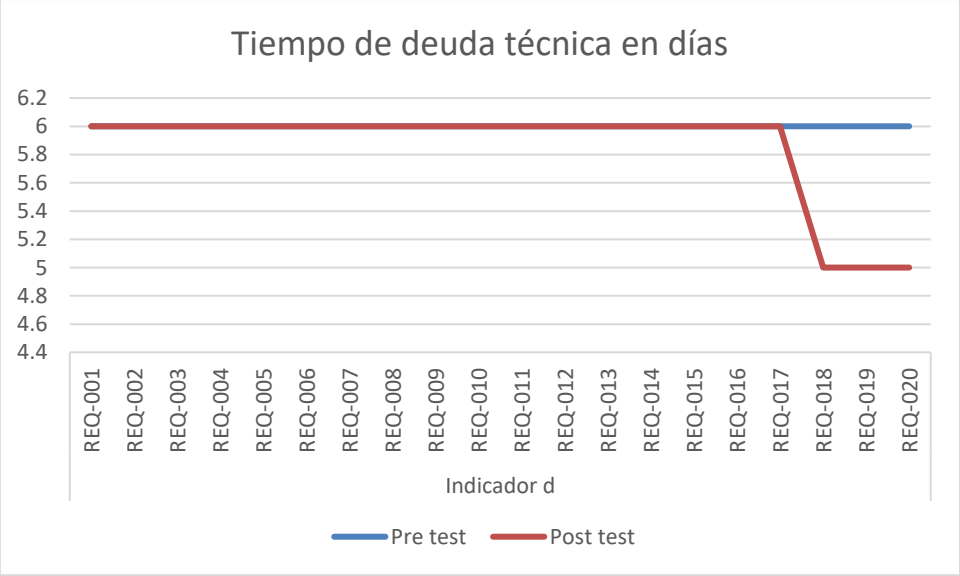


Figura 9. Comparación de tiempo de deuda técnica en días de Pre test y Post test

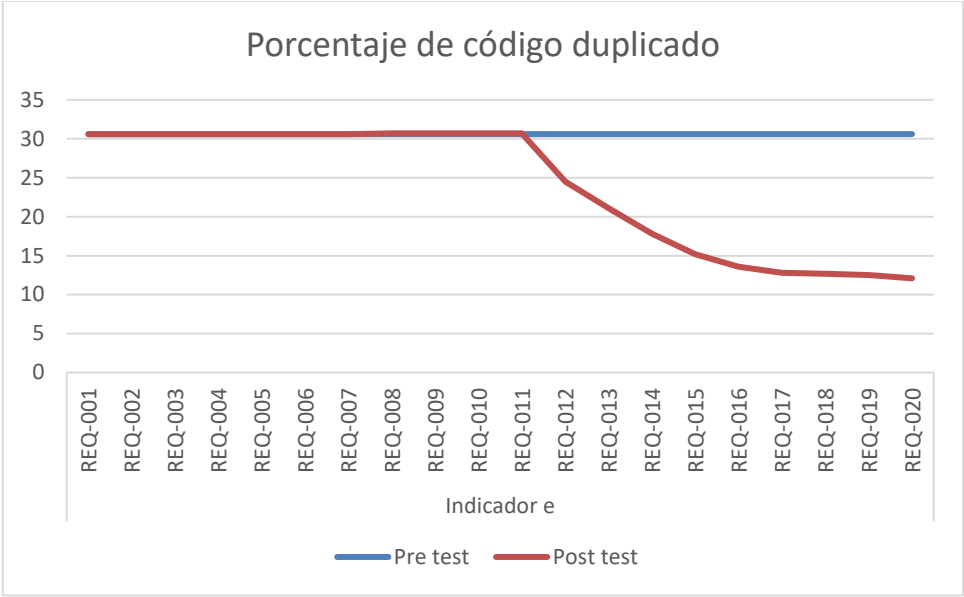


Figura 10. Comparación de porcentaje de código duplicado de Pre test y Post test

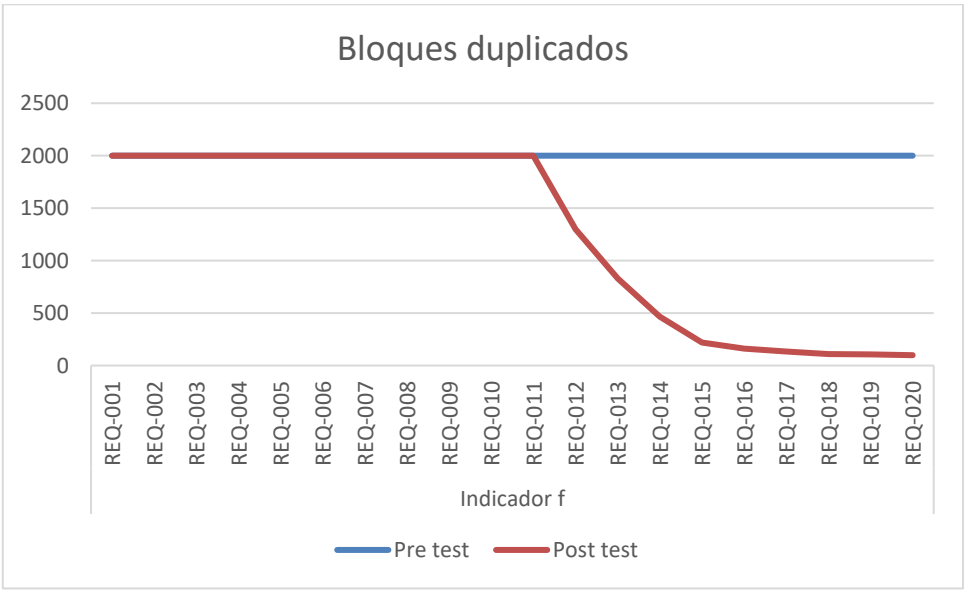


Figura 11. Comparación de bloques duplicados de Pre test y Post test

Objetivo Específico 3

Determinar la influencia de la implementación del continuous delivery pipeline en la automatización de despliegue de un software basado en Python.

En la Tabla 7 se muestran los indicadores de agilidad y calidad del proceso de desarrollo de la variable dependiente presentes en el instrumento ubicado en el Anexo 3, cada una denotada por letras, todos los indicadores fueron tomados de la operacionalización de variables (ver Anexo 1).

Tabla 7 *Indicadores de agilidad y calidad del proceso de desarrollo*

	Indicadores
g	Tiempo de solución de requerimiento en minutos
h	Tiempo de integración de código en segundos
i	Número de errores al realizar despliegue
j	Tiempo que toma realizar despliegue en minutos

Al igual que en el objetivo anterior se recolectó información por medio del instrumento validado, el cual se muestra en el Anexo 3, el orden en el que se realizó la resolución de requerimientos también fue el mismo, es decir: REQ-001, REQ-002, REQ-003, REQ-004, REQ-005, REQ-006, REQ-007, REQ-008, REQ-009, REQ-010, REQ-011, REQ-012, REQ-013, REQ-014, REQ-015, REQ-016, REQ-017, REQ-018, REQ-019, REQ-020.

PRE-TEST

El pre-test consistió en la resolución de 20 requerimientos, los cuales se muestran en el Anexo 6, la resolución de los requerimientos se dio de la manera en la que se trabaja actualmente (sin análisis de calidad de código y con despliegues manuales), es decir sin el continuous delivery pipeline. El procedimiento consistió en resolver el requerimiento y realizar el despliegue manualmente (ver Tabla 8).

Tabla 8 *Resultado pre-test de indicadores de agilidad y calidad del proceso de desarrollo*

	Indicadores			
Requerimiento	**g**	**h**	**i**	**j**
REQ-001	19,00	20,00	0	4,00
REQ-002	20,00	20,00	0	5,00
REQ-003	20,00	21,00	0	6,00
REQ-004	21,00	21,00	0	7,00
REQ-005	22,00	21,00	0	7,00
REQ-006	22,00	21,00	0	7,00
REQ-007	23,00	22,00	0	8,00
REQ-008	25,00	23,00	0	8,00
REQ-009	26,00	24,00	0	8,00
REQ-010	28,00	26,00	0	10,00
REQ-011	30,00	28,00	0	13,00
REQ-012	25,00	22,00	0	14,00
REQ-013	10,00	35,00	0	10,00
REQ-014	10,00	24,00	0	12,00
REQ-015	13,00	21,00	0	9,00
REQ-016	10,00	22,00	0	9,00
REQ-017	10,00	22,00	0	8,00
REQ-018	11,00	24,00	0	10,00
REQ-019	15,00	32,00	0	10,00
REQ-020	10,00	21,00	0	10,00

Fuente: Instrumento guía de observación.

Interpretación

- **Indicador g - Tiempo de solución de requerimiento en minutos**

  El tiempo promedio de solución de requerimientos es de 18.75, el mínimo es de 10 y el máximo es de 30 minutos.

- **Indicador h - Tiempo de integración de código en segundos**

  El tiempo promedio de integración de código es de 23.50, el mínimo es de 20 y el máximo es de 35 segundos.

- **Indicador i - Número de errores al realizar despliegue**

  Se puede apreciar que el número de errores al realizar despliegues manualmente se mantuvo en cero lo que significa que realizando despliegues manuales no siempre se

generan errores; sin embargo, cuando existen procesos manuales los errores van a ocurrir tarde o temprano.

- **Indicador j - Tiempo que toma realizar despliegue en minutos**

  El tiempo promedio que toma realizar despliegues es de 8.75, el mínimo es de 4 y el máximo es de 14 minutos.

POST-TEST

El post-test consistió en la resolución de 20 requerimientos, los cuales se muestran en el Anexo 6, la resolución de los requerimientos se dio con el uso del continuous delivery pipeline, lo que permitió realizar análisis de calidad de código y despliegues automatizados. El procedimiento consistió en resolver el requerimiento, luego se realiza un análisis de calidad de código con ayuda de SonarQube se mejora la calidad de código un promedio de tiempo de 8.2 minutos y con ayuda de Bamboo se realiza el despliegue automatizado (ver Tabla 9).

Tabla 9 *Resultado post-test de indicadores de agilidad y calidad del proceso de desarrollo*

	Indicadores			
Requerimiento	**g**	**h**	**i**	**j**
REQ-001	40,00	52,00	0	0,58
REQ-002	37,00	45,00	0	0,56
REQ-003	34,00	44,00	0	0,56
REQ-004	33,00	44,00	0	0,56
REQ-005	32,00	44,00	0	0,73
REQ-006	31,00	45,00	0	0,70
REQ-007	30,00	42,00	0	0,70
REQ-008	28,00	52,00	0	0,70
REQ-009	28,00	47,00	0	0,58
REQ-010	26,00	46,00	0	0,56
REQ-011	24,00	45,00	0	0,73
REQ-012	35,00	44,00	0	0,73
REQ-013	20,00	57,00	0	1,00
REQ-014	16,00	41,00	0	0,71
REQ-015	19,00	41,00	0	0,75
REQ-016	19,00	40,00	0	1,00
REQ-017	20,00	40,00	0	1,00

REQ-018	17,00	45,00	0	0,73
REQ-019	25,00	51,00	0	0,73
REQ-020	20,00	39,00	0	0,78

Fuente: Instrumento guía de observación.

Interpretación

- **Indicador g - Tiempo de solución de requerimiento en minutos**

  El tiempo promedio de solución de requerimientos es de 26.70, el mínimo es de 16 y el máximo es de 40 minutos.

- **Indicador h - Tiempo de integración de código en segundos**

  El tiempo promedio de integración de código es de 45.20, el mínimo es de 39 y el máximo es de 57 segundos.

- **Indicador i - Número de errores al realizar despliegue**

  Se puede apreciar que el número de errores al realizar despliegues automatizados se mantuvo en cero lo que significa que realizando despliegues automatizados se disminuye la probabilidad de que ocurran errores, además en caso se generen errores se podrá realizar un rollback en menos de un minuto.

- **Indicador j - Tiempo que toma realizar despliegue en minutos**

  El tiempo promedio que toma realizar despliegues es de 0.72, el mínimo es de 0.56 y el máximo es de 1 minuto.

Resultado

Comparando los resultados del pre-test y post-test.

- **Indicador g - Tiempo de solución de requerimiento en minutos**

  Se puede apreciar que el tiempo de solución de requerimiento ha aumentado, esto se debe a que el tiempo de solución del requerimiento del post test es la suma del tiempo que tomó solucionar un requerimiento más el tiempo que tomó mejorar la calidad de código que en promedio fue de 8.2 minutos. El aumento de este indicador es temporal, ya que

una vez que el software tenga calidad alta, entonces este tiempo disminuirá, porque el tiempo de mejora de calidad será menor por lo que se considera que el continuous delivery pipeline mejora los resultados de este indicador (ver Figura 12).

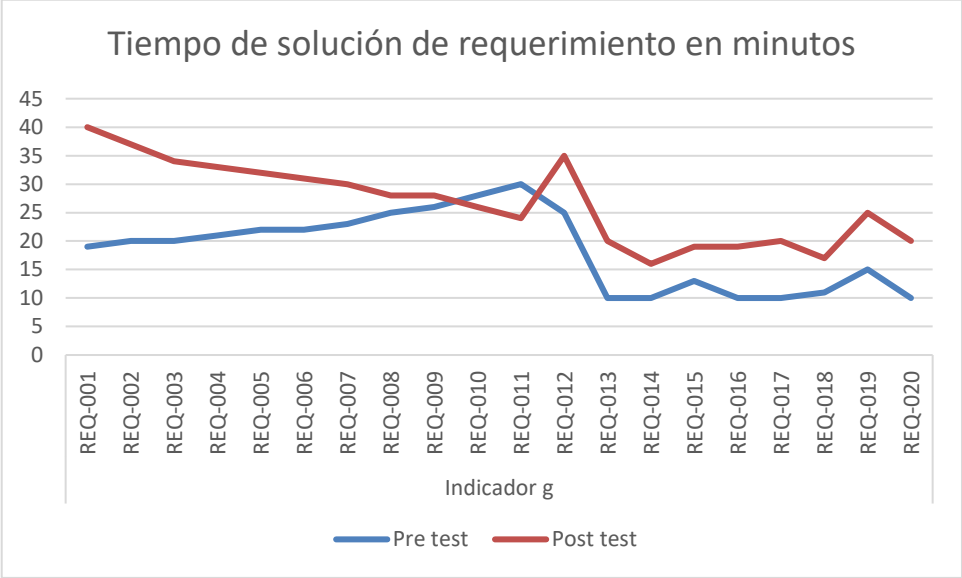


Figura 12. Comparación de tiempo de solución de requerimiento en minutos de Pre test y Post test

- **Indicador h - Tiempo de integración de código en segundos**

  Se puede apreciar que el tiempo de integración de código ha aumentado, esto se debe a que este tiempo es la suma del tiempo que tomó integrar el código dos veces: la primera fue cuando se solucionó el requerimiento y la segunda cuando se mejoró la calidad de código. El aumento de este indicador es temporal, ya que una vez que el software tenga calidad alta, entonces no se tendrá que mejorar la calidad a menos que los cambios introducidos en el código para la solución de los requerimientos sea código con calidad baja, por lo que se considera que el continuous delivery pipeline mejora los resultados de este indicador (ver Figura 13).

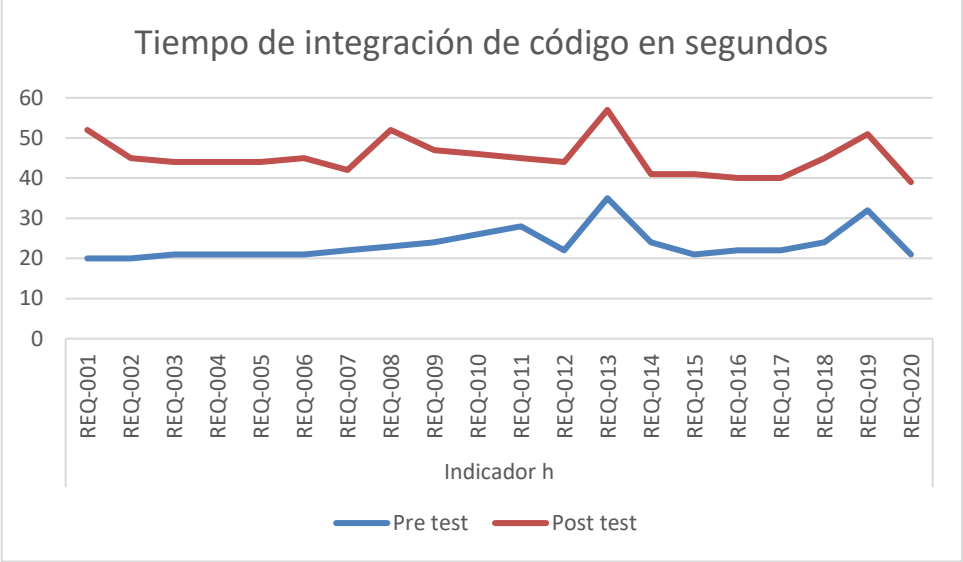


Figura 13. Comparación de tiempo de integración de código en segundos de Pre test y Post test

- **Indicador i - Número de errores al realizar despliegue**

  Se puede apreciar que el número de errores al realizar despliegues se mantuvo en cero lo que significa que al usar el continuous delivery pipeline o al no usarlo no se generaron errores; sin embargo, por el hecho de que el continuous delivery pipeline automatiza el proceso de despliegue entonces la probabilidad de que los errores ocurran es menor comparado con hacer despliegues manualmente, por lo que se considera que el continuous delivery pipeline mejora los resultados de este indicador.

- **Indicador j - Tiempo que toma realizar despliegue en minutos**

  Con respecto a este indicador se puede apreciar que los tiempos han disminuido, por lo que se considera que el continuous delivery pipeline mejora los resultados de este indicador (ver Figura 14).

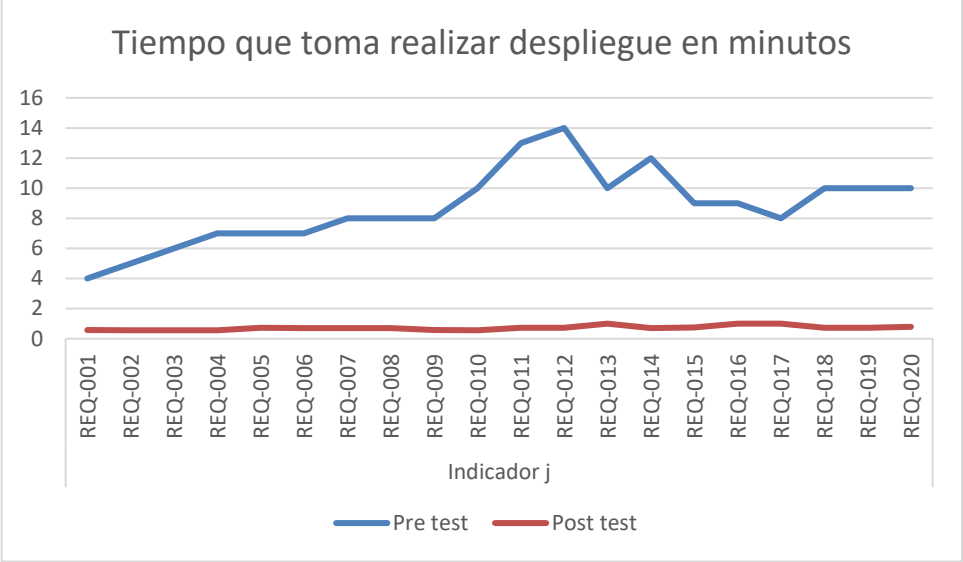


Figura 14. Comparación de tiempo que toma realizar despliegue en minutos de Pre test y Post test

Prueba contrastación de hipótesis con Wilcoxon

Indicador – Número de errores

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el número de errores de un software basado en Python.
- H_1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el número de errores de un software basado en Python.

En la Figura 16 se muestran los resultados de la prueba de Wilcoxon al indicador: Número de errores.

Interpretación

P-Valor = 0.000, este valor es menor que 0.05, entonces se acepta la hipótesis alterna: La implementación de un continuous delivery pipeline influye positivamente en el número de errores de un software basado en Python.

Indicador – Número de vulnerabilidades

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el número de vulnerabilidades de un software basado en Python.
- H_1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el número de vulnerabilidades de un software basado en Python.

En la Figura 17 se muestran los resultados de la prueba de Wilcoxon al indicador: Número de vulnerabilidades.

Interpretación

P-Valor = 0.005, este valor es menor que 0.05, entonces se acepta la hipótesis alterna: La implementación de un continuous delivery pipeline influye positivamente en el número de vulnerabilidades de un software basado en Python.

Indicador - Número de code smells

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el número de code smells de un software basado en Python.
- H_1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el número de code smells de un software basado en Python.

En la Figura 18 se muestran los resultados de la prueba de Wilcoxon al indicador: Número de code smells.

Interpretación

P-Valor = 0.000, este valor es menor que 0.05, entonces se acepta la hipótesis alterna: La implementación de un continuous delivery pipeline influye positivamente en el número de code smells de un software basado en Python.

Indicador - Tiempo de deuda técnica

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el tiempo de deuda técnica de un software basado en Python.
- H_1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el tiempo de deuda técnica de un software basado en Python.

En la Figura 19 se muestran los resultados de la prueba de Wilcoxon al indicador: Tiempo de deuda técnica.

Interpretación

P-Valor = 0.000, este valor es menor que 0.05, entonces se acepta la hipótesis alterna: La implementación de un continuous delivery pipeline influye positivamente en el tiempo de deuda técnica de un software basado en Python.

Indicador - Porcentaje de código duplicado

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el porcentaje de código duplicado de un software basado en Python.
- H_1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el porcentaje de código duplicado de un software basado en Python.

En la Figura 20 se muestran los resultados de la prueba de Wilcoxon al indicador: Porcentaje de código duplicado.

Interpretación

P-Valor = 0.013, este valor es menor que 0.05, entonces se acepta la hipótesis alterna: La implementación de un continuous delivery pipeline influye positivamente en el porcentaje de código duplicado de un software basado en Python.

Indicador – Bloques duplicados

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el número de bloques duplicados de un software basado en Python.
- H_1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el número de bloques duplicados de un software basado en Python.

En la Figura 21 se muestran los resultados de la prueba de Wilcoxon al indicador: Bloques duplicados.

Interpretación

P-Valor = 0.008, este valor es menor que 0.05, entonces se acepta la hipótesis alterna: La implementación de un continuous delivery pipeline influye positivamente en el número de bloques duplicados de un software basado en Python.

Indicador - Tiempo de solución de requerimiento en minutos

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el tiempo de solución de requerimientos de un software basado en Python.
- H_1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el tiempo de solución de requerimientos de un software basado en Python.

En la Figura 22 se muestran los resultados de la prueba de Wilcoxon al indicador: Tiempo de solución de requerimiento en minutos.

Interpretación

P-Valor = 0.000, este valor es menor que 0.05, entonces se acepta la hipótesis alterna: La implementación de un continuous delivery pipeline influye positivamente en el tiempo de solución de requerimientos de un software basado en Python.

Indicador - Tiempo de integración de código en segundos

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el tiempo de integración de código de un software basado en Python.
- H_1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el tiempo de integración de código de un software basado en Python.

En la Figura 23 se muestran los resultados de la prueba de Wilcoxon al indicador: Tiempo de integración de código en segundos.

Interpretación

P-Valor = 0.000, este valor es menor que 0.05, entonces se acepta la hipótesis alterna: La implementación de un continuous delivery pipeline influye positivamente en el tiempo de integración de código de un software basado en Python.

Indicador - Tiempo que toma realizar despliegue en minutos

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el tiempo que toma realizar despliegues de un software basado en Python.

- H_1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el tiempo que toma realizar despliegues de un software basado en Python.

En la Figura 24 se muestran los resultados de la prueba de Wilcoxon al indicador: Tiempo que toma realizar despliegue en minutos.

Interpretación

P-Valor = 0.000, este valor es menor que 0.05, entonces se acepta la hipótesis alterna: La implementación de un continuous delivery pipeline influye positivamente en el tiempo que toma realizar despliegues de un software basado en Python.

Indicador – Número de errores al realizar despliegues

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el número de errores al realizar despliegues de un software basado en Python.
- H1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el número de errores al realizar despliegues de un software basado en Python.

Interpretación

Para el indicador número de errores al realizar despliegues; no se realizó la prueba de contrastación de hipótesis, ya que los resultados del pre-test y post-test no cambiaron, es decir la diferencia entre los valores del post-test y pre-test es cero.

La implementación de un continuous delivery pipeline influye positivamente en el proceso de despliegue de requerimientos de un software.

Hipótesis:

- H_0: P-Valor > 0.05 → La implementación de un continuous delivery pipeline no influye positivamente en el proceso de despliegue de requerimientos de un software.
- H1: P-Valor <= 0.05 → La implementación de un continuous delivery pipeline influye positivamente en el proceso de despliegue de requerimientos de un software.

Interpretación

Dado que se aceptaron todas las hipótesis específicas entonces se acepta la hipótesis general: La implementación de un continuous delivery pipeline influye positivamente en el proceso de despliegue de requerimientos de un software.

CAPÍTULO IV. DISCUSIÓN Y CONCLUSIONES

4.1. Discusión

Una actividad del ciclo de desarrollo de software que toma mucho tiempo, es el despliegue, como menciona Stelligent (2017), donde concluyó que con la implementación de un continuous delivery pipeline con Amazon Web Services se pueden realizar despliegues con tan solo hacer clic en un botón, con referencia a lo mencionado lo comprobé, ya que para que se inicie cada despliegue al ambiente simulado de producción con el uso del continuous delivery pipeline consistía en hacer clic en un botón, el tiempo promedio de despliegue fue de 0.72 minutos para tener la aplicación funcionando con la solución de los requerimientos en el ambiente simulado de producción.

En base a los resultados obtenidos se acepta lo mencionado por Farley y Humble (2010), quienes concluyeron que entre mayor es la calidad de código de un proyecto software, entonces la facilidad de realizar mantenimiento aumenta; ya que, conforme el indicador code smell pasaba de 488 a 397 el tiempo de solución de requerimientos disminuyó a 10 minutos en la solución del último requerimiento. Esto se evaluó con ayuda de la herramienta SonarQube, la cual mide calidad de código.

Finalmente cabe mencionar que en contrastación con McFadden (2017), donde concluyó que con la implementación de un continuous delivery pipeline se reduce el tiempo de despliegue como también el tiempo de ciclo de desarrollo de software, con referencia a ello comprobé que el tiempo de despliegue sí se reduce notablemente, ya que el tiempo promedio de despliegue manual fue de 8.75 minutos y el tiempo promedio de despliegue automatizado fue de 0.72 minutos.

4.2. Conclusiones

La implementación del continuous delivery pipeline influye positivamente en el proceso de despliegue de requerimientos de un software, ya que se comprobó su influencia con la prueba de contrastación de hipótesis con Wilcoxon, además se considera positiva ya que todos los valores de los indicadores disminuyeron, por ejemplo, el número de errores pasó de 191 a 39, por lo tanto, se aprueba el objetivo general.

Se logró implementar el continuous delivery pipeline para probar el software basado en Python con la metodología Kanban, la cual permitió mapear todas las actividades necesarias para desarrollar la investigación.

Se logró determinar la influencia de la implementación del continuous delivery pipeline en la calidad de código del proyecto portal de empleo, siendo esta una influencia positiva en la calidad de código, ya que en la resolución de 20 requerimientos, el número de errores disminuyó en 78.92%, número de vulnerabilidades en 100%, número de code smells en 18.98%, tiempo de deuda técnica en días en 16.67%, porcentaje de código duplicado en 60.46%, bloques duplicados en 95.05%; lo cual conlleva a que la realización del mantenimiento del proyecto portal de empleo se facilite conforme la calidad de código del proyecto aumenta.

Se logró determinar la influencia de la implementación del continuous delivery pipeline en la automatización de despliegues del proyecto portal de empleo, siendo esta una influencia positiva en la mejora del proceso de despliegue, ya que el tiempo

promedio de despliegue manual fue de 8.75 minutos y el tiempo promedio de despliegue automatizado fue de 0.72 minutos.

REFERENCIAS

Atlassian. (2018). *Bamboo Support*. Obtenido de https://confluence.atlassian.com/bamboo

Atlassian. (2018). *Evaluator Resources Support*. Obtenido de https://confluence.atlassian.com/confeval/development-tools-evaluator-resources/bitbucket/bitbucket-what-is-bitbucket

aws. (2018). *aws*. Obtenido de https://aws.amazon.com/es/what-is-aws/

Campbell, A. (30 de Mayo de 2017). *SonarQube Documentation.* Obtenido de https://docs.sonarqube.org/display/SONAR/Issues

Cecil, R. (2009). *Clean Code.* Prentice Hall.

Chen, L. (2016). *Continuous Delivery: Overcoming Adoption Obstacles.* Irlanda.

Chen, L. (2017). *Buy In SonarQube as A Quality Gate for Test Automation*. Obtenido de https://blogs.perficient.com/2017/04/13/buy-in-sonarqube-as-a-quality-gate-for-test-automation/

Cobanoglu, C. (2016). *Adopting Continuous Delivery Practices.*

Farley, D., & Humble, J. (2010). *Continuous Delivery.* Boston: Pearson Education, Inc.

Gaudin, O. (21 de Noviembre de 2016). *Sonarqube concepts.* Obtenido de https://docs.sonarqube.org/display/SONAR/Concepts

Gmeiner, J., & Haslinger, J. (2015). *Automated Testing in the Continuous Delivery Pipeline: A Case Study of an Online Company.*

Holland, B. (22 de Febrero de 2012). *Sitepoint*. Obtenido de https://www.sitepoint.com/how-to-define-a-software-tester/

Kim, G. (13 de febrero de 2014). *IT REVOLUTION*. Obtenido de https://itrevolution.com/the-amazing-devops-transformation-of-the-hp-laserjet-firmware-team-gary-gruver/

Laptick, S. (20 de Octuble de 2016). *sitepoint*. Obtenido de https://www.sitepoint.com/how-why-to-use-the-kanban-methodology-for-software-development/

Mahanta, P., Adige, V., Pole, A., & Rajkumar. (2016). *DevOps Culture and its impact on Cloud Delivery.*

Mäntylä, M., & Vanhanen, J. (2010). *Software Deployment Activities and Challenges – A Case Study of Four Software Product Companies.*

McFadden, C. (6 de Febrero de 2017). *Our DevOps Journey: Achieving Continuous Delivery and Improvement.* Obtenido de https://www.sparkpost.com/blog/continuous-delivery-deployment-2/

Meng, Wegman, Zhang, Chen, & Chafle. (2017). *IT troubleshooting with drift.*

Morelos, E. (22 de Agosto de 2018). *Enterpreneur*. Obtenido de https://www.entrepreneur.com/article/304376

Nduka Oyom, A. (2017). *Understanding the MVC pattern in Django*. Obtenido de https://medium.com/shecodeafrica/understanding-the-mvc-pattern-in-django-edda05b9f43f

Parizo, C. (2018). *TechBeacon*. Obtenido de https://techbeacon.com/continuous-delivery-only-web-companies

Perera, P., Bandara, M., & Indika, P. (2016). *Evaluating the Impact of DevOps Practice in Sri.* Department of Computer Science & Engineering, University of Moratuwa.

Python. (2017). *Python.* Obtenido de https://www.python.org/about/

Raphael, D. (31 de 05 de 2018). *What is each server*. Obtenido de https://www.gratasoftware.com/what-is-each-server-for-dev-test-uat-or-staging-demo-and-production/

Roche, J. (2013). *Adopting DevOps Practices in Quality Assurance.*

Rouse, M. (2007). *TechTarget*. Obtenido de https://searchsoftwarequality.techtarget.com/definition/build

Rouse, M. (Marzo de 2008). *TechTarget*. Obtenido de https://searchsoftwarequality.techtarget.com/definition/release

salesforce. (2017). *Development Lifecycle Guide.*

Sampieri, R. (2014). *Metodología de la investigación.*

Savor, T., Williams, L., Douglas, M., Beck, K., Stumm, M., & Gentili, M. (2017). *Continuous deployment at Facebook and OANDA.* ResearchGate.

Shahina, M., Babara, A., & Zhub, L. (2015). *Continuous Integration, Delivery and Deployment: A Systematic.* Australia.

SonarQube. (2018). Obtenido de SonarQube Documentation: https://docs.sonarqube.org/7.4/

sonarsource. (2018). *sonarsouce Python Rules* . Obtenido de https://rules.sonarsource.com/python

stelligent. (2017). *Sony Pictures Technologies Develops DevOps Solution with Stelligent to*. Obtenido de https://s3.amazonaws.com/stelligent_casestudies/stelligent_sony.pdf

Subramanian, N. (2017). *The Software Deployment Process and Automation.* Texas.

Swersky, D. (31 de Mayo de 2018). *RAYGUN*. Obtenido de https://raygun.com/blog/software-development-life-cycle/#sdlc-infographic

Wider, A., & Deger, C. (1 de Marzo de 2017). *Getting Smart: Applying Continuous Delivery to Data Science to Drive Car Sales.* Obtenido de https://www.thoughtworks.com/es/insights/blog/getting-smart-applying-continuous-delivery-data-science-drive-car-sales

Xiaoqing, F. (2009). *Software quality function deployment.*

Zhu, L., Xu, D., Tran, A., Xu, X., Bass, L., Weber, I., & Dwarakanathan, S. (2015). *Achieving Reliable High Frequency Releases in Cloud Environments.* Australia.

ANEXOS

Anexo 1 Operacionalización de variables

Operacionalización de variable independiente

VARIABLES	DEFINICIÓN CONCEPTUAL	DIMENSIONES	SUBDIMEN-SIONES	INDICADORES
Implementación de un continuous delivery pipeline	La implementación de un continuous delivery pipeline permite llevar a cabo la práctica ágil denominada continuous delivery, cuyo objetivo es asegurarse que una aplicación siempre esté lista para que se despliegue a ambientes de producción (Shahina, Babara, & Zhub, 2015).	Continuous integration	Control de versiones	Tiempo de creación de un build en minutos. Cantidad de líneas de código por build. Tiempo de ejecución de tests automatizados. Porcentaje de disponibilidad del software.
		Despliegues automatizados	Respuesta a necesidades de negocio	Tiempo de despliegue después de un commit en minutos. Tiempo de retroalimentación del cliente en minutos. Número de correcciones de defectos por despliegue. Tiempo que toma volver a versiones anteriores en minutos.

Operacionalización de variable dependiente

VARIABLES	DEFINICIÓN CONCEPTUAL	DIMENSIONES	SUBDIME N-SIONES	INDICADORES	HERRAMIENTA
Proceso de despliegue	El despliegue de un software es el último paso en el ciclo de desarrollo de software, después del despliegue la organización cliente se verá beneficiada por usar el software instalado o actualizado (Subramanian, 2017).	Despliegue	Agilidad	Número de errores al realizar despliegue. Tiempo de despliegue en minutos.	Bamboo
		Calidad	Calidad de software	Número de errores. Número de vulnerabilidades. Número de code smells. Tiempo de deuda técnica en días. Porcentaje de código duplicado. Bloques duplicados.	SonarQube
			Calidad del proceso de desarrollo	Tiempo de entrega de software en horas. Tiempo de integración de código en segundos.	Bamboo

Anexo 2 Fichas de validación de instrumento

UNIVERSIDAD PRIVADA DEL NORTE

FICHA PARA VALIDACION DEL INSTRUMENTO

I. REFERENCIA

1.1. Experto: Luis Mendoza Escalante
1.2. Especialidad: Ingeniería de Sistemas
1.3. Cargo actual: Ingeniero Software
1.4. Grado académico: Ingeniero de Sistemas
1.5. Institución: b2b2b
1.6. Tipo de instrumento: Ficha de registro de Datos
1.7. Lugar y fecha: Cajamarca, 1 Julio 2014

II. TABLA DE VALORACION POR EVIDENCIAS

N°	EVIDENCIAS	VALORACION					
		5	4	3	2	1	0
1	Pertinencia de indicadores	✓					
2	Formulado con lenguaje apropiado	✓					
3	Adecuado para los sujetos en estudio		✓				
4	Facilita la prueba de hipótesis		✓				
5	Suficiencia para medir la variable	✓					
6	Facilita la interpretación del instrumento	✓					
7	Acorde al avance de la ciencia y tecnología	✓					
8	Expresado en hechos perceptibles	✓					
9	Tiene secuencia lógica	✓					
10	Basado en aspectos teóricos	✓					
	Total	40	8				

Coeficiente de valoración porcentual: c = 48

III. OBSERVACIONES Y/O RECOMENDACIONES

.....
.....
.....
.....
.....

Firma y sello del Experto

UNIVERSIDAD PRIVADA DEL NORTE

FICHA PARA VALIDACION DEL INSTRUMENTO

I. REFERENCIA

1.1. Experto: Luis Miguel Cotrina Malca
1.2. Especialidad: Ingeniero de Sistemas
1.3. Cargo actual: Gerente de Proyectos
1.4. Grado académico: Master en Project Management
1.5. Institución: Datcos
1.6. Tipo de instrumento: Ficha de Registro de Datos
1.7. Lugar y fecha: Cajamarca, 30 de junio del 2019.

II. TABLA DE VALORACION POR EVIDENCIAS

N°	EVIDENCIAS	VALORACION					
		5	4	3	2	1	0
1	Pertinencia de indicadores	✓					
2	Formulado con lenguaje apropiado	✓					
3	Adecuado para los sujetos en estudio	✓					
4	Facilita la prueba de hipótesis		✓				
5	Suficiencia para medir la variable	✓					
6	Facilita la interpretación del instrumento	✓					
7	Acorde al avance de la ciencia y tecnología	✓					
8	Expresado en hechos perceptibles	✓					
9	Tiene secuencia lógica	✓					
10	Basado en aspectos teóricos	✓					
	Total	45	4				

Coeficiente de valoración porcentual: c = 49

III. OBSERVACIONES Y/O RECOMENDACIONES

Enlazar el instrumento con la hipotesis y el objetivo de aplicarlo.

Firma y sello del Experto

UNIVERSIDAD PRIVADA DEL NORTE

FICHA PARA VALIDACION DEL INSTRUMENTO

I. REFERENCIA

1.1. Experto: JAVIER ORTIZ
1.2. Especialidad: ING. INFORMATICO
1.3. Cargo actual: GERENTE I&D
1.4. Grado académico: SUPERIOR
1.5. Institución: PUCP - BIT2BIT
1.6. Tipo de instrumento: FICHA DE REGISTRO DE DATOS
1.7. Lugar y fecha: CAJAMARCA, 3 DE JULIO 2017

II. TABLA DE VALORACION POR EVIDENCIAS

N°	EVIDENCIAS	VALORACION					
		5	4	3	2	1	0
1	Pertinencia de indicadores		✓				
2	Formulado con lenguaje apropiado		✓				
3	Adecuado para los sujetos en estudio	✓					
4	Facilita la prueba de hipótesis	✓					
5	Suficiencia para medir la variable		✓				
6	Facilita la interpretación del instrumento	✓					
7	Acorde al avance de la ciencia y tecnología		✓				
8	Expresado en hechos perceptibles			✓			
9	Tiene secuencia lógica	✓					
10	Basado en aspectos teóricos	✓					
	Total						

Coeficiente de valoración porcentual: c = 44

III. OBSERVACIONES Y/O RECOMENDACIONES

REVISAR NOMENCLATURA UTILIZADA EN MEJORA DE PROCESOS (LEAN/KANBAN) PARA TIEMPOS A MEDIR (TIEMPO DE CICLO)

Firma
JAVIER O. ORTIZ OSSA
Coordinador General del Proyecto
BIT2BIT S.A.C.

Anexo 3 Instrumento proceso de despliegue

GUÍA DE OBSERVACIÓN - PROCESO DE DESPLIEGUE

Encargado de infraestructura: _____

Fecha: _____

Proyecto software a evaluar: _____

Código de requerimiento: _____

VARIABLE: PROCESO DE DESPLIEGUE	
INDICADORES	**RESULTADO**
Número de errores al realizar despliegue.	
Tiempo que toma realizar despliegue en minutos.	
Número de errores.	
Número de vulnerabilidades.	
Número de code smells.	
Tiempo de deuda técnica en días.	
Porcentaje de código duplicado.	
Bloques duplicados.	
Tiempo de solución de requerimiento en minutos.	
Tiempo de integración de código en segundos.	

Anexo 4 Confiabilidad del instrumento con Alpha de Cronbach

Estadísticas de fiabilidad

Alfa de Cronbach	N de elementos
,719	10

Estadísticas de total de elemento

	Media de escala si el elemento se ha suprimido	Varianza de escala si el elemento se ha suprimido	Correlación total de elementos corregida	Alfa de Cronbach si el elemento se ha suprimido
NroErrores	582,8727	71,218	,000	,728
NroVulnerabilidades	766,8727	71,218	,000	,728
NroCodeSmells	277,8727	71,218	,000	,728
TiempoDeudaTecnica	761,8727	71,218	,000	,728
PorcentajeCodigoDuplicado	737,2727	71,218	,000	,728
BloquesDuplicados	765,8727	71,218	,000	,728
TiempoSolucionRequerimiento	744,6000	24,400	,975	,547
TiempoIntegracionCòdigoFuente	745,4182	34,764	,978	,527
NroErroresDespliegue	767,8727	71,218	,000	,728
TiempoDespliegue	760,3273	37,218	,951	,542

Figura 15. Confiabilidad del instrumento con Alpha de Cronbach
Fuente: software IBM SPSS Statistics versión 23

Anexo 5 Estadísticos Wilcoxon

Prueba de rangos con signo de Wilcoxon

Rangos

		N	Rango promedio	Suma de rangos
Después - Antes	Rangos negativos	20[a]	10,50	210,00
	Rangos positivos	0[b]	,00	,00
	Empates	0[c]		
	Total	20		

a. Después < Antes

b. Después > Antes

c. Después = Antes

Estadísticos de prueba[a]

	Después - Antes
Z	-3,920[b]
Sig. asintótica (bilateral)	,000

a. Prueba de rangos con signo de Wilcoxon

b. Se basa en rangos positivos.

Figura 16. Prueba Wilcoxon a indicador número de errores
Fuente: software IBM SPSS Statistics versión 23

Prueba de rangos con signo de Wilcoxon

Rangos

		N	Rango promedio	Suma de rangos
Después - Antes	Rangos negativos	8[a]	4,50	36,00
	Rangos positivos	0[b]	,00	,00
	Empates	12[c]		
	Total	20		

a. Después < Antes

b. Después > Antes

c. Después = Antes

Estadísticos de prueba[a]

	Después - Antes
Z	-2,828[b]
Sig. asintótica (bilateral)	,005

a. Prueba de rangos con signo de Wilcoxon

b. Se basa en rangos positivos.

Figura 17. Prueba Wilcoxon a indicador número de vulnerabilidaes
Fuente: software IBM SPSS Statistics versión 23

Prueba de rangos con signo de Wilcoxon

Rangos

		N	Rango promedio	Suma de rangos
Después - Antes	Rangos negativos	20[a]	10,50	210,00
	Rangos positivos	0[b]	,00	,00
	Empates	0[c]		
	Total	20		

a. Después < Antes

b. Después > Antes

c. Después = Antes

Estadísticos de prueba[a]

	Después - Antes
Z	-3,920[b]
Sig. asintótica (bilateral)	,000

a. Prueba de rangos con signo de Wilcoxon

b. Se basa en rangos positivos.

Figura 18. Prueba Wilcoxon a indicador número de code smells
Fuente: software IBM SPSS Statistics versión 23

Prueba de rangos con signo de Wilcoxon

Rangos

		N	Rango promedio	Suma de rangos
Después - Antes	Rangos negativos	3[a]	2,00	6,00
	Rangos positivos	0[b]	,00	,00
	Empates	17[c]		
	Total	20		

a. Después < Antes

b. Después > Antes

c. Después = Antes

Estadísticos de prueba[a]

	Después - Antes
Z	-1,732[b]
Sig. asintótica (bilateral)	,000

a. Prueba de rangos con signo de Wilcoxon

b. Se basa en rangos positivos.

Figura 19. Prueba Wilcoxon a indicador tiempo de deuda técnica
Fuente: software IBM SPSS Statistics versión 23

Prueba de rangos con signo de Wilcoxon

Rangos

		N	Rango promedio	Suma de rangos
Después - Antes	Rangos negativos	9[a]	9,00	81,00
	Rangos positivos	4[b]	2,50	10,00
	Empates	7[c]		
	Total	20		

a. Después < Antes

b. Después > Antes

c. Después = Antes

Estadísticos de prueba[a]

	Después - Antes
Z	-2,489[b]
Sig. asintótica (bilateral)	,013

a. Prueba de rangos con signo de Wilcoxon

b. Se basa en rangos positivos.

Figura 20. Prueba Wilcoxon a indicador porcentaje de código duplicado
Fuente: software IBM SPSS Statistics versión 23

Prueba de rangos con signo de Wilcoxon

Rangos

		N	Rango promedio	Suma de rangos
Después - Antes	Rangos negativos	9[a]	5,00	45,00
	Rangos positivos	0[b]	,00	,00
	Empates	11[c]		
	Total	20		

a. Después < Antes

b. Después > Antes

c. Después = Antes

Estadísticos de prueba[a]

	Después - Antes
Z	-2,666[b]
Sig. asintótica (bilateral)	,008

a. Prueba de rangos con signo de Wilcoxon

b. Se basa en rangos positivos.

Figura 21. Prueba Wilcoxon a indicador número de bloques duplicados
Fuente: software IBM SPSS Statistics versión 23

Prueba de rangos con signo de Wilcoxon

Rangos

		N	Rango promedio	Suma de rangos
Después - Antes	Rangos negativos	2[a]	3,50	7,00
	Rangos positivos	18[b]	11,28	203,00
	Empates	0[c]		
	Total	20		

a. Después < Antes

b. Después > Antes

c. Después = Antes

Estadísticos de prueba[a]

	Después - Antes
Z	-3,674[b]
Sig. asintótica (bilateral)	,000

a. Prueba de rangos con signo de Wilcoxon

b. Se basa en rangos negativos.

Figura 22. Prueba Wilcoxon a indicador tiempo de solución de requerimiento en minutos
Fuente: software IBM SPSS Statistics versión 23

Prueba de rangos con signo de Wilcoxon

Rangos

		N	Rango promedio	Suma de rangos
Después - Antes	Rangos negativos	0[a]	,00	,00
	Rangos positivos	20[b]	10,50	210,00
	Empates	0[c]		
	Total	20		

a. Después < Antes

b. Después > Antes

c. Después = Antes

Estadísticos de prueba[a]

	Después - Antes
Z	-3,927[b]
Sig. asintótica (bilateral)	,000

a. Prueba de rangos con signo de Wilcoxon

b. Se basa en rangos negativos.

Figura 23. Prueba Wilcoxon a indicador tiempo de integración de código en segundos
Fuente: software IBM SPSS Statistics versión 23

Prueba de rangos con signo de Wilcoxon

Rangos

		N	Rango promedio	Suma de rangos
Después - Antes	Rangos negativos	20[a]	10,50	210,00
	Rangos positivos	0[b]	,00	,00
	Empates	0[c]		
	Total	20		

a. Después < Antes

b. Después > Antes

c. Después = Antes

Estadísticos de prueba[a]

	Después - Antes
Z	-3,921[b]
Sig. asintótica (bilateral)	,000

a. Prueba de rangos con signo de Wilcoxon

b. Se basa en rangos positivos.

Figura 24. Prueba Wilcoxon a indicador tiempo que toma realizar despliegue en minutos
Fuente: software IBM SPSS Statistics versión 23

Anexo 6 Requerimientos de prueba

Tabla 10 *Requerimientos de prueba*

CÓDIGO	TÍTULO	Comentarios
REQ-001	Agregar dropdownlist en las reglas de accesos de la aplicación.	El dropdownlist debe permitir escoger tipos posibilitando así generar reglas.
REQ-002	Configuración de cron.	Agregar tarea cron que verifique si se eliminó un registro de la base de datos.
REQ-003	Título de la aplicación se muestra intermitentemente dependiendo a la concurrencia de usuarios	Título en la página de log in no se muestra.
REQ-004	Placeholder de log in no es correcto.	Verificar porque el placeholder de log in no es correcto.
REQ-005	Autenticación errónea para usuario administrador no muestra mensaje correcto.	Verificar error reportado
REQ-006	Reglas de acceso no están funcionando.	Usuario N00029972 ingresa siempre, sin importar que las reglas de acceso no lo deben permitir.
REQ-007	Usuario N00029972 no ingresa, pero debe ingresar debido a sus datos.	Usuario N00029972 no puede ingresar a la aplicación
REQ-008	Mostrar 5 opciones en la página de inicio, sección para contratistas.	Verificar error reportado
REQ-009	Página principal de la aplicación no es responsiva.	Hacer responsiva la página principal de la aplicación
REQ-010	Autenticación errónea no muestra mensaje correcto error.	Verificar lo reportado
REQ-011	Agregar imágenes a los reportajes que se encuentran en la página de inicio.	Validar archivos estáticos
REQ-012	Cambiar otros requisitos al final del formulario de solicitar practicantes	Posicionar “otros requisitos” al final de solicitud de practicantes.
REQ-013	Otros requisitos al final del formulario de solicitud de egresados.	Otros requisitos al final del formulario de solicitud de egresados.
REQ-014	Agregar funcionalidad de generación de cv automáticamente	Verificar requerimiento
REQ-015	Validar inconsistencia de créditos de estudiantes	Variación de datos de reportes
REQ-016	Cambiar eliminar Idioma y agregar módulo de empresas.	Validar lo solicitado
REQ-017	Eliminar niveles no funciona	Validar lo solicitado
REQ-018	Cambiar idiomas de reporte después de conocimientos de informática.	Posicionar sección idiomas de reporte después de la sección de conocimientos de informática.
REQ-019	Agregar funcionalidad de usuario bloqueados	Crear una funcionalidad que permita agregar usuarios bloqueados, estos usuarios no tendrán acceso a la aplicación.

REQ-020	Agregar al siguiente usuario como bloqueado N00029973	Este usuario no debe poder ingresar a la aplicación

Anexo 7 Documentos desarrollo Proyecto

CONTINUOUS DELIVERY PIPELINE
Plan del Proyecto

Introducción

En el presente proyecto se desarrollará un continuous delivery pipeline, el cual automatizará el proceso de despliegue y posibilitará realizar análisis de calidad de código.

El proyecto será desarrollado utilizando la metodología Kanban siguiendo sus tres principios: visualiza producción, limita el trabajo en progreso y medir tiempo tomado en tareas.

Problema/Necesidad del negocio a ser resuelta

En la actualidad el proceso de despliegue de requerimientos del proyecto portal de empleo es totalmente manual lo que produce errores y mucho tiempo en su ejecución, además no se analiza la calidad de código, lo cual disminuye la productividad en la solución de requerimientos del proyecto portal de empleo.

Mediante la implementación del continuous delivery pipeline, se propone automatizar el proceso de despliegue y posibilitar el análisis de calidad de código.

Antecedentes y descripción de la situación actual y esperada.

Stelligent (2017), en su investigación cuyo objetivo fue realizar despliegues rápidos en lugar de esperar días o semanas hasta que se siga un proceso manual de despliegue en la compañía Advent, la cual es una compañía que fabrica software diseñado para automatizar la contabilidad de carteras para firmas de administración de inversiones, en esta publicación se afirma que un proceso de despliegue manual es lento y alarga el ciclo de desarrollo de software. Como conclusión se implementó un continuous delivery pipeline con tecnología Amazon Web Services logrando así realizar

despliegues con tan solo hacer clic en un botón.

El proceso de despliegue de requerimientos a los ambientes de test, uat y producción del proyecto portal de empleo son totalmente manuales, provocando que el proceso tome un tiempo promedio de 30 minutos, causando así que la aplicación no funcione durante todo ese proceso.

La situación deseada es contar con un continuous delivery pipeline para que disminuya el tiempo de realización de despliegue.

Viabilidad

Técnica

- El proyecto es viable ya que, se podrá utilizar en diversos lenguajes de programación, también existirá la posibilidad de cambiar los componentes del pipeline con otros, como por ejemplo cambiar la herramienta de integración continua Bamboo por Jenkins si se desea.
- El proyecto es viable porque se cuenta con conocimientos básicos en infraestructura y desarrollo de software.

Económica

El desarrollo del proyecto es viable económicamente, por lo que no demandará muchos gastos económicos como por ejemplo la compra de un servidor físico para realizar pruebas, en su lugar considero adoptar Amazon Web Services (AWS).

Justificación del Proyecto

Esta investigación servirá para futuros trabajos, en las cuales se puedan determinar las variables tratadas en este proyecto como implementación de continuous delivery y proceso de despliegue e incentiva a otros investigadores a solucionar problemas similares, dependiendo del tipo de proyecto de software.

La presente investigación servirá a muchas empresas grandes y pequeñas desarrolladoras de software que enfrentan problemas al realizar despliegues, permitiendo así que se realicen a tiempo, sin fallas y con posibilidad de realizar rollbacks en pocos minutos. Cabe recalcar que, con cambios a los planes de integración y despliegue de Bamboo el continuous delivery pipeline funcionará para un lenguaje de programación específico que se requiera, además la implementación del continuous delivery pipeline se podrá llevar a cabo con diferentes herramientas no solo con las presentadas en esta investigación.

Interesados y Colaboradores.

Tabla 11 *Interesados y colaboradores del proyecto*

Nombre	Descripción	Responsabilidades
Paredes Atencio, Richard Manuel	Equipo de Trabajo	Diseñador del pipeline Implementador del pipeline
Uceda Martos, Patricia	Asesor	Asesorar el desarrollo de plan de tesis

Descripción General del Proyecto

Continuous delivery pipeline automatizará el proceso de despliegues y realizará análisis de calidad de código. El pipeline servirá para diversos lenguajes de programación y también posibilita agregar o cambiar las herramientas del continous delivery pipeline, por ejemplo, se podrá cambiar la herramienta de integración continua Bitbucket por Github.

Declaración de Trabajo.

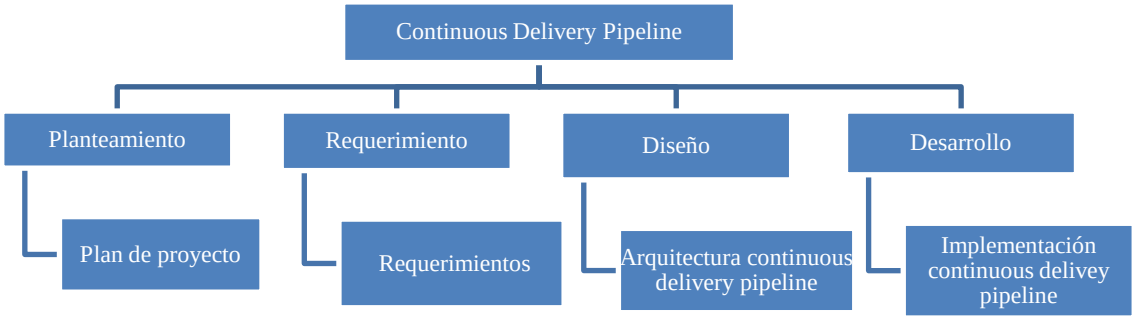


Figura 25. Etapas para desarrollar el proyecto

Objetivos del Proyecto.

- Determinar la influencia de un continuous delivery pipeline en el proceso de despliegue de requerimientos de un software.
- Implementar el continuous delivery pipeline para probar un software basado en Python con la metodología Kanban.
- Determinar la influencia de la implementación del continuous delivery pipeline en la calidad de código de un software basado en Python.
- Determinar la influencia de la implementación del continuous delivery pipeline en la automatización de despliegue de un software basado en Python.

Enfoque del proyecto

El presente proyecto nace de haber experimentado la realización de despliegues de requerimientos totalmente manual. Por ello he planteado desarrollar un continuous delivery pipeline que ayude a resolver este problema y así poder agilizar este proceso y mejorar la calidad del software. Para el desarrollo del proyecto mencionado se usa la metodología Kanban.

El desarrollo del proyecto está basado en investigaciones ya realizadas en diversas empresas a nivel mundial. La implementación del continuous delivery pipeline se

desarrollará aprovechando la tecnología Amazon Web Services. Al final del proyecto tendremos un continuos delivey pipeline que automatizará el proceso de despliegue y permita el análisis de calidad de código.

Presupuesto

(ver Tabla 2)

Organización del Proyecto

Tabla 12 *Organización del proyecto*

Integrante del Equipo	Roles que desempeña
Paredes Atencio, Richard	Autor y desarrollador del proyecto

CONTINUOUS DELIVERY PIPELINE
Requerimientos del proyecto

Tabla 13 *Listado de requerimientos del proyecto*

Descripción del requerimiento	**Documento Referencia**
El continuous delivey pipeline debe permitir realizar análisis de calidad de código.	Anexo 12
El continuous delivey pipeline debe permitir realizar despliegue automatizado.	Anexo 13
El continuous delivey pipeline debe permitir realizar rollback automatizado.	Anexo 14
Implementación del continuous delivey pipeline.	Anexo 8 Anexo 9 Anexo 10 Anexo 11

CONTINUOUS DELIVERY PIPELINE
Descripción de la arquitectura del continuous delivery pipeline

Propósito

El presente documento proporcionará una visión general del continuous delivery pipeline implementado con ayuda de Amazon Web Services (AWS).

Objetivo

- Diseñar un continuous delivery pipeline para tener una mejor visión al desarrollar el proyecto.

Continuous delivey pipeline es un conjunto de procesos automatizados que permiten a los desarrolladores y encargados de operaciones a compilar y desplegar código a ambientes como producción.

Características: Su diseño cuenta con las siguientes herramientas:

- Repositorio de código fuente.
- Herramienta de integración continua.
- Analizador de calidad de código

Diseño del continuous delivey pipeline

(ver Figura 5)

Descripción

Desarrollador de software

Desarrollador de software que subirá cambios de código al repositorio de código fuente.

Repositorio de código fuente

Representa una herramienta que permite el control de código fuente en el desarrollo de cualquier aplicación software.

Herramienta de integración continua.

Herramienta que permite la creación de planes de integración y despliegue, conformados por tareas que contienen las configuraciones como por ejemplo compilación, análisis de calidad de código y despliegue.

Analizador de calidad de código.

Herramienta que realiza el análisis de calidad de código, dando como resultados indicadores como por ejemplo deuda técnica.

Continuos delivery pipeline con las herramientas consideradas en la implementación (ver Figura 26)

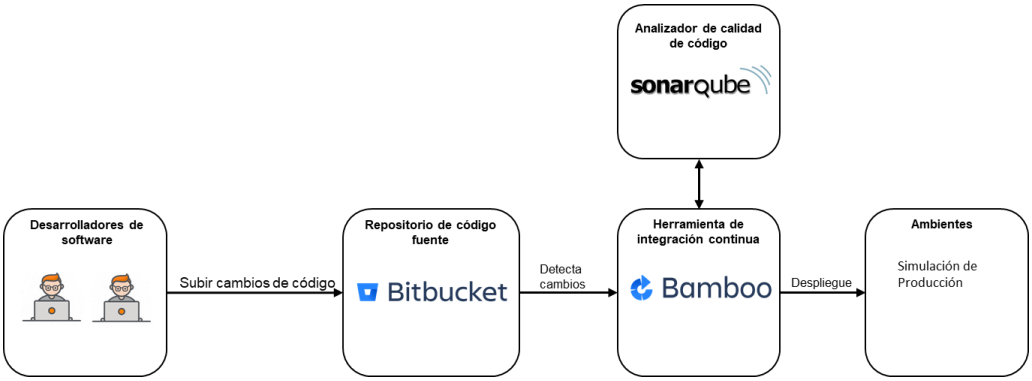


Figura 26. Continuous delivery pipeline implementado

CONTINUOUS DELIVERY PIPELINE

Proceso del continuous delivery pipeline

El proceso del continuous delivery pipeline mostrado en la Figura 26, es el siguiente

1. Desarrollador de software

Un desarrollador de software, programa la solución de un requerimiento, luego sube todos los cambios de código fuente realizados al repositorio de código fuente Bitbucket.

2. Repositorio de código fuente - Bitbucket

El código fuente del repositorio en Bitbucket está actualizado según todos los cambios que los desarrolladores de software hayan subido.

3. Herramienta de integración continua - Bamboo

Bamboo tiene un plan de integración configurado que detecta automáticamente todos los cambios en el repositorio de código fuente Bitbucket, luego se realiza el análisis de calidad de código en SonarQube automáticamente con los nuevos cambios en el repositorio de código fuente Bitbucket, dando como resultado un build.

4. Analizador de calidad de código – SonarQube

El análisis de calidad de código es realizado tomando en cuenta el código fuente considerado en la ejecución del build.

5. Herramienta de despliegue

Una vez que se haya ejecutado el plan de integración, entonces se puede realizar un despliegue automatizado al ambiente de producción cuando se requiera tomando en cuenta el código fuente considerado en la ejecución del build.

Anexo 8 Instalación de Bamboo

Precondiciones

- Tener instalado PostgreSQL.

```
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib
```

- Instalar Java en Linux

1. Acceder al servidor Linux y ejecutar los siguientes comandos.

```
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
```

2. Luego ejecutar el siguiente comando para instalar la versión 8.

```
sudo apt-get install oracle-java8-installer
```

3. Ejecutar el siguiente comando con la ruta de instalación.

```
JAVA_HOME="/usr/lib/jvm/java-8-oracle"
```

- Verificar que la versión de Java este correctamente instalado, realizando lo siguiente:

a. Revisar que la versión de Java sea la correcta, con el siguiente comando:

```
java -version
```

b. Revisar que la variable de entorno, apunte o contenga el directorio donde está instalado el JDK:

```
echo $JAVA_HOME
```

c. Ejecutar el siguiente comando y copiar la línea al final del archivo. JAVA_HOME="/usr/lib/jvm/java-8-oracle"

```
sudo nano /etc/environment
```

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games$
JAVA_HOME="/usr/lib/jvm/java-8-oracle"
```

Figura 27. Ejemplo configuración Java
Fuente: Servidor AWS

Nota: para acceder a Bamboo desde el navegador es necesario abrir el puerto 8085 en el servidor.

Instalación de Bamboo 5.14.3.1

La seguridad de Bamboo es muy importante, para lo cual se recomienda seguir los consejos de seguridad de la siguiente.

1. Ingresar a https://www.atlassian.com/software/bamboo/download y hacer clic en Linux.
2. Copiar el enlace del botón *Descarga* (Download) de **Bamboo 5.14.3.1 - TAR.GZ Archive**.
3. Descargar el archivo usando el enlace del paso anterior, al momento de la creación del manual es el 5.14.3.1.

```
wget . https://www.atlassian.com/software/bamboo/downloads/binary/atlassian-bamboo-5.14.3.1.tar.gz
```

4. Crear la carpeta que contendrá los archivos.

```
mkdir /opt/atlassian
```

5. Ejecutar el siguiente comando desde la carpeta en donde se encuentra el instalador, con la ejecución del comando se pasará el archivo descomprimido a la carpeta /opt/atlassian

```
tar xvf atlassian-bamboo-5.14.3.1.tar.gz -C /opt/atlassian/
```

6. Acceder a la carpeta Atlassian y renombrar la carpeta de bamboo.

```
cd /opt/atlassian
mv atlassian-bamboo-5.14.3.1 bamboo
```

7. Crear las carpetas que contendrán Bamboo Home (esta debe ser diferente a la carpeta de instalación).

```
mkdir /var/atlassian
mkdir /var/atlassian/application-data/
mkdir /var/atlassian/application-data/bamboo
```

8. Modificar el archivo bamboo-init.properties para que indique la ruta de Bamboo Home.

```
nano /opt/atlassian/bamboo/atlassian-bamboo/WEB-INF/classes/bamboo-init.properties
```

9. Remover el numeral (#) delante de la variable bamboo.home y modificar el valor de esta para que sea la ruta creada en el paso 7.

```
bamboo.home=/var/atlassian/application-data/bamboo
```

10. Crear un usuario dedicado a Bamboo, el cual se encargará de ejecutar el servicio.

    Nota: --create-home: crear directorio home del usuario, -c : comentario.

```
sudo useradd --create-home -c "Bamboo role account" bamboo
```

11. Otorgar permisos al usuario creado sobre las carpetas de Bamboo.

```
sudo chown -R bamboo: /opt/atlassian/bamboo
sudo chown -R bamboo: /var/atlassian/application-data/bamboo
```

12. Crear el archivo bamboo en /etc/init.d/.

```
sudo nano /etc/init.d/bamboo
```

13. Dentro del archivo copiar lo siguiente:

```
#!/bin/sh
set -e
### BEGIN INIT INFO
# Provides: bamboo
# Required-Start: $local_fs $remote_fs $network $time
# Required-Stop: $local_fs $remote_fs $network $time
# Should-Start: $syslog
# Should-Stop: $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Atlassian Bamboo Server
### END INIT INFO
# INIT Script
#####################################
```

```
# Define some variables
# Name of app ( bamboo, Confluence, etc )
APP=bamboo
# Name of the user to run as
USER=bamboo
# Location of application's bin directory
BASE=/opt/atlassian/bamboo

case "$1" in
  # Start command
  start)
    echo "Starting $APP"
    /bin/su -  $USER  -c "export  BAMBOO_HOME=${BAMBOO_HOME};  $BASE/bin/startup.sh  &> /dev/null"
    ;;
  # Stop command
  stop)
    echo "Stopping $APP"
    /bin/su - $USER -c "$BASE/bin/shutdown.sh &> /dev/null"
    echo "$APP stopped successfully"
    ;;
  # Restart command
  restart)
      $0 stop
      sleep 5
      $0 start
      ;;
  *)
    echo "Usage: /etc/init.d/$APP {start|restart|stop}"
    exit 1
    ;;
esac


exit 0
```

14. Dar permiso de ejecutable.

```
sudo chmod a+x /etc/init.d/bamboo
```

15. Actualizar el registro de servicios.

```
sudo update-rc.d bamboo defaults
```

16. Iniciar el servicio e ingresar a Bamboo con el siguiente enlace http://ip-del-servidor:8085

```
sudo service bamboo start
```

Anexo 9 Instalación de SonarQube

Requisitos:

Antes de proceder con los pasos del presente manual, debe tener instalado lo siguiente:

- Oracle JDK 1.8.X
- PostgreSQL 8.x o 9.x
- Abrir el puerto 9000 del servidor

Instalación

1. Descargue el instalador de la versión LTS de SonarQube:

```
wget --no-check-certificate https://sonarsource.bintray.com/Distribution/sonarqube/sonarqube-5.6.4.zip
```

2. Descomprimir el instalador en la carpeta */opt/*:

```
unzip sonarqube-5.6.4.zip -d /opt/
```

3. Crear el esquema en la base de datos PostgreSQL y cambiar la contraseña de postgres:

```
sudo -u postgres psql
ALTER USER postgres PASSWORD 'newpassword';
sudo su postgres
psql
CREATE USER "sonarqube" WITH PASSWORD 's0n4rqub!';
CREATE DATABASE "sonarqube" WITH ENCODING 'UTF-8' LC_COLLATE = 'en_US.UTF-8' LC_CTYPE = 'en_US.UTF-8';
GRANT ALL PRIVILEGES ON DATABASE "sonarqube" TO "sonarqube";
```

4. Actualizar el archivo ubicado en *<install_directory>/conf/sonar.properties* para que utilice la base de datos creada previamente:

```
sonar.jdbc.username=sonarqube
sonar.jdbc.password=s0n4rqub!
sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube
```

5. Para iniciar SonarQube, ingrese el siguiente comando:

   a) Ingrese al siguiente directorio:

```
cd /opt/sonarqube-5.6.4/bin/linux-x86-64
```

b) Ejecutar el siguiente archivo:

```
sudo ./sonar.sh start
```

c) Ingresar a SonarQube utilizando el siguiente enlace: http://ip-del-server:9000

Anexo 10 Configuración de análisis de calidad de código

Manual de integración de Bamboo con SonarQube, los siguientes comandos serán ejecutados en el servidor dónde está instalado Bamboo.

1. Descargar comprimido.

```
wget http://repo1.maven.org/maven2/org/codehaus/sonar/runner/sonar-runner-dist/2.4/sonar-runner-dist-2.4.zip
```

2. Descomprimir archivo en la ruta deseada.

```
unzip home/ubuntu/sonar-runner-dist-2.4.zip  -d /opt/
```

3. Ingresar a la carpeta descomprimida y actualizar la configuración global modificando el archivo **/opt/conf/sonar-runner.properties.** o **/opt/sonar-runner/conf/ sonar-runner.properties**

```
mv sonar-runner-2.4/ sonar-runner
cd /opt/sonar-runner
#----- Default SonarQube server
sonar.host.url=http://localhost:9000/sonar
```

4. Agregar el directorio **bin** de la carpeta de instalación de **sonar-runner** al **path** del SO.

```
PATH=$PATH:/opt/sonar-runner/bin/
```

5. Verificar la instalación básica.

```
sonar-runner -h
```

Manual de Análisis Estático con SonarQube

Prerrequisitos

- Tener instalado Sonar-Runner

- Instalar en SonarQube los plugins de los lenguajes que se deseen examinar. Para instalar los plugins de los lenguajes en SonarQube: ingresar a SonarQube, luego ir a Administration → System → Update Center.

1. Clonar repositorio Bitbucket
2. Agregar el siguiente archivo (**sonar-project.properties**) en la ruta principal del repositorio clonado:

```
sonar.projectKey=PythonProject:branch
sonar.projectName=My Project
sonar.projectVersion=1.0
sonar.language=py
sonar.sources=.
sonar.sourceEncoding=UTF-8
```

- sonar.projectKey, es el key del proyecto creado en SonarQube. (el projectKey recién se lo define acá; es decir se puede poner cualquier Project key conservando ":branch".
- sonar.projectName, es el nombre del proyecto creado en SonarQube (este proyecto se crea automáticamente).
- sonar.language, lenguaje de programación que se va realizar el análisis.

3. Subir el cambio usando la herramienta de gestión de código fuente.
4. Creación de Build Plan, un plan está contenido dentro de un proyecto de Build.
    1. Ingresar a Bamboo con una cuenta de Administrador.
    2. Hacer clic en el menú **Create**, luego en **Create a new plan**.
    3. Se visualiza el formulario de **Create a new plan**.

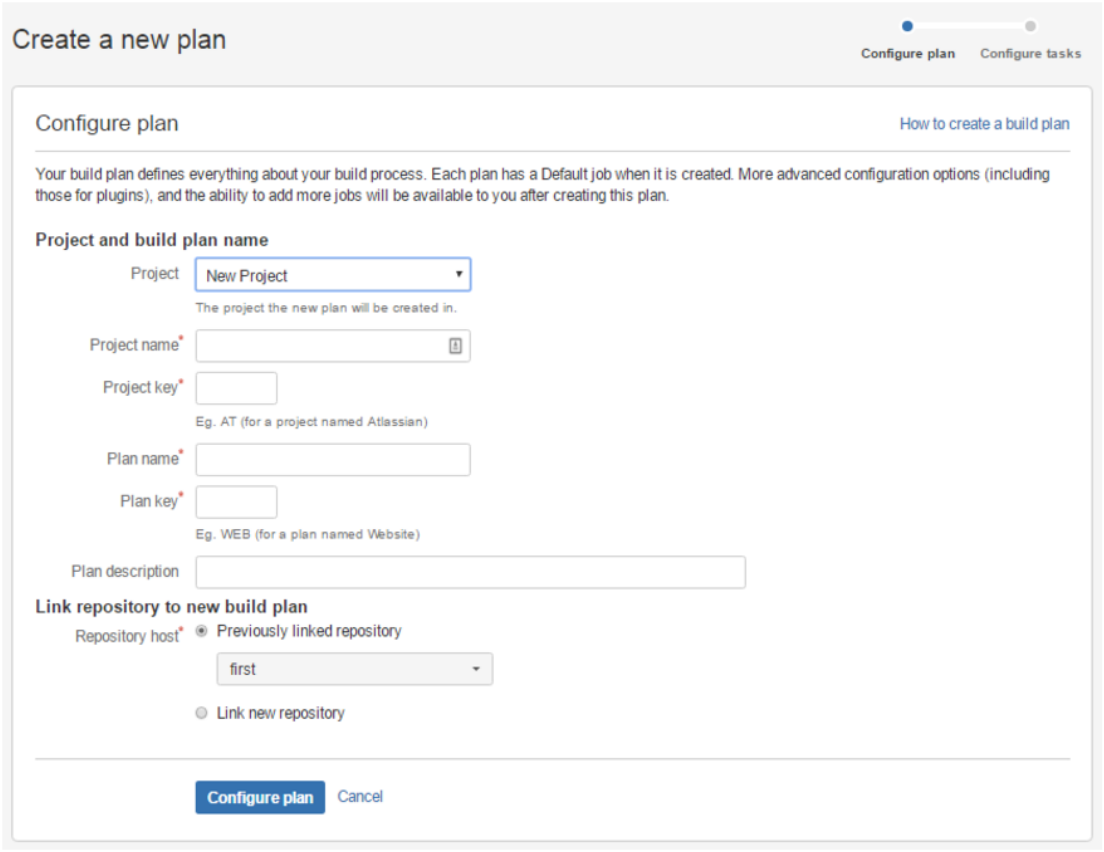


Figura 28. Crear nuevo plan de integración
Fuente: Bamboo

4. Ingresar los siguientes datos:

    a. Project.

        i. New Project, si se desea crear un proyecto de build. Al seleccionar **New Project**, se debe ingresar el nombre y key del nuevo proyecto.

        ii. Existing projects. si se desea agregar el plan de buid a un proyecto existente.

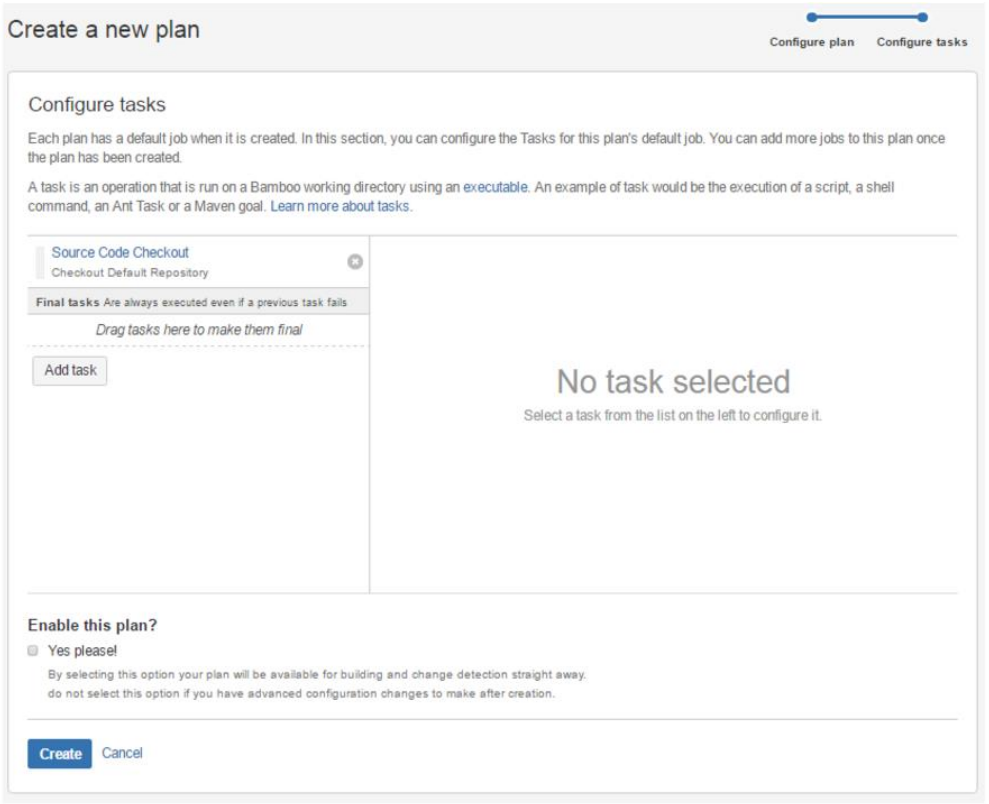
    b. Plan name.

    c. Plan key.

d. Repository host, elegir la opción **Previously linked repository** y seleccionar un repositorio linkeado previamente.

5. Elegir **Configure plan**.

6. Se visualiza el formulario de **Configure tasks**.



Fiaura 29. Configuración de tareas de plan de integración
Fuente: Bamboo

7. Agregar tareas.

   a. **Source Code Checkout**, por cada repositorio a analizar. Configurar los campos:

      i. Repository.

      ii. Checkout Directory: ingresar un nombre que identifica al repositorio.

   b. **Script.** Ingresar el siguiente contenido:

i. Task Description, ingrese la descripción de la tarea.

ii. Interpreter, seleccionar la opción de *bin/sh or cmd.exe*.

iii. Script location, seleccionar la opción de *inline*.

iv. Script Body, ingresar la siguiente sección de código.

```
sed -i 's/branch/{newValue}/g' sonar-project.properties

sh /opt/sonar-runner/bin/sonar-runner
```

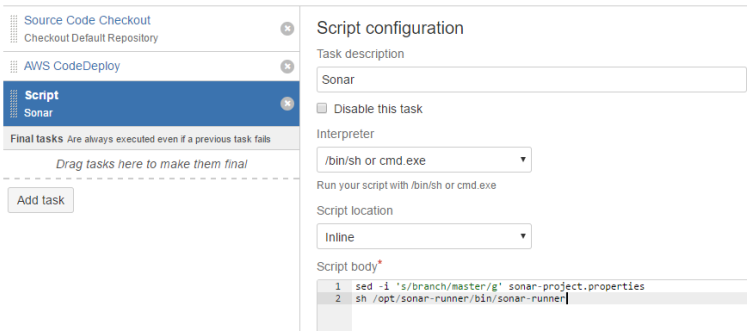


Figura 30. Configuración de tarea análisis de calidad de código en plan de integración
Fuente: Bamboo

8. Si se desea ejecutar el plan por primera vez, marcar la opción **Yes please**.
9. Elegir **Create**.
10. Se visualiza la ventana **Plan Configuration**.
11. Hacer clic en **Default job**.
12. Dentro del Job, hacer clic en el tab **Artifacts**.
13. Seleccionar **create definition**.
14. Ingresar los datos:
    a. Name.
    b. Location, dejar valor por defecto.
    c. Copy pattern: **/**.**

d. Marcar la opción **Shared**.

e. Hacer clic en **Create**.

15. En la parte superior derecha, hacer clic en **Run → Run plan**.

16. Hacer clic en el botón de **Save**.

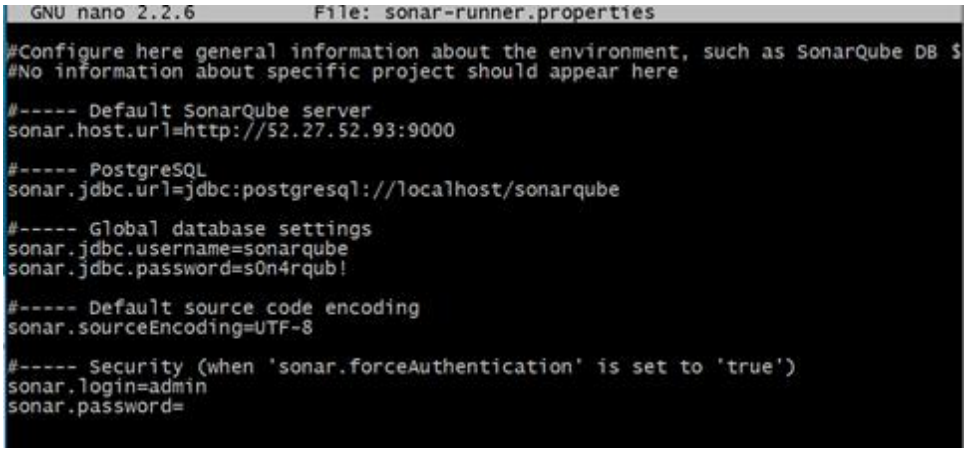


Figura 31. Archivo de configuración SonarQube
Fuente: Servidor AWS

Anexo 11 Configuración despliegue automatizado

1. Ingresar a Bamboo.
2. Hacer clic en el menú **Create deployment Project.**
3. Ingresar un nombre de proyecto.
4. Seleccionar un **Build plan.**
5. Hacer clic en agregar ambiente.
6. Ingresar nombre de ambiente.
7. Se muestra la siguiente interfaz gráfica.

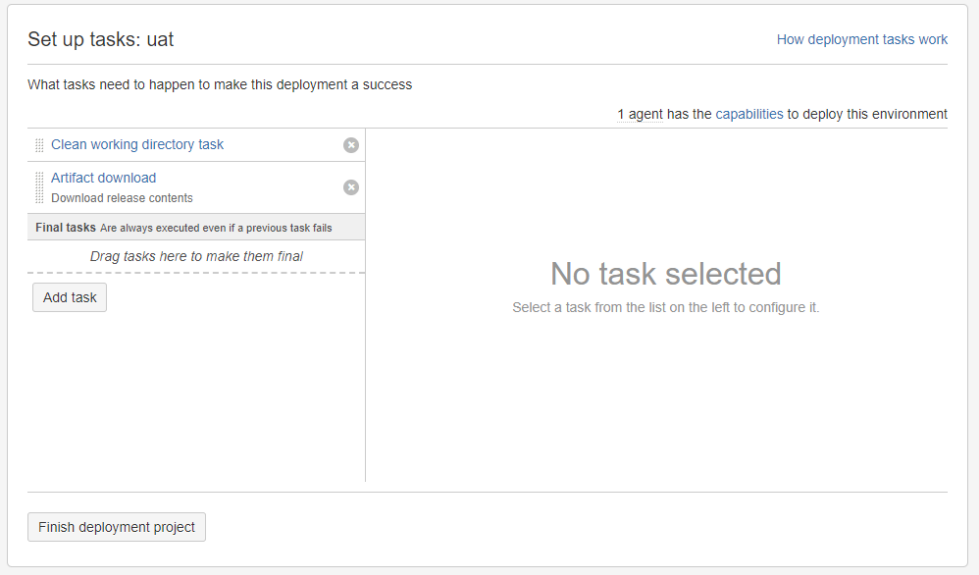


Figura 32. Crear plan de despliegue
Fuente: Bamboo

8. Agregar las siguientes tareas
9. Agregar tarea de tipo ssh
    a. Ingresar el siguiente comando

    cd /home/ubuntu/scriptsForPostTest

    . killProcessPort8000.sh

10. Agregar tarea de tipo ssh

    a. Ingresar el siguiente comando

    cd /home/ubuntu/scriptsForPostTest

    . RemoveAndCreateDir.sh

11. Agregar tarea de tipo ssh

    a. Ingresar el siguiente comando

    cd /home/ubuntu/scriptsForPostTest

    . RemovePythonlog.sh

12. Agregar tarea de tipo scp

    a. Seleccionar artefacto e ingresar remote path

13. Agregar tarea de tipo ssh

    a. Ingresar el siguiente comando

    cd /home/ubuntu/scriptsForPostTest

    . RestartNGIXActivateEnvStartGunicorn.sh

14. A continuación, el contenido de los scripts:

    a. KillProcessPort8000.sh

```
#!/bin/bash
echo "STARTING CLEAN DEPLOY"
echo "kill process"
sudo lsof -t -i tcp:8000 | xargs kill -9
```

Figura 33. Script KillProcessPort8000
Fuente: Servidor AWS

    b. RemoveAndCreateDir.sh

```
cd /home/ubuntu/
sudo rm -r project
sudo mkdir project
```

Figura 34. Script RemoveAndCreateDir

Fuente: Servidor AWS

c. RemovePythonlog.sh

```
sudo chmod -R 777 /tmp/python.log
```

Figura 35. Script RemovePythonlog.sh
Fuente: Servidor AWS

d. RestartNGIXActivateEnvStartGunicorn.sh

```
#!/bin/bash
echo "STARTING CLEAN DEPLOY"
echo "kill process port 8000 again"
sudo lsof -t -i tcp:8000 | xargs kill -9
echo "UPDATING SOURCES"
echo "RESTARTING NGINX"
sudo service nginx restart
sleep 5s
echo "ACTIVATING VIRTUAL ENV"
cd /home/ubuntu/
source env/bin/activate
echo "ACTIVATING GUNICORN"
cd /home/ubuntu/project
nohup gunicorn project.wsgi:application --bind 0.0.0.0:8000 </dev/null>/dev/null 2>&1 &
echo "GUNICORN STARTED"
```

Figura 36. Script RestartNginx ActivateEnvStartGunicorn
Fuente: Servidor AWS

15. Si se desea ejecutar el plan por primera vez, marcar la opción **Yes please**.

16. Hacer clic en Create.

Anexo 12 Caso de uso Subir cambios de código y realizar análisis de calidad de código

Especificación del Caso de Uso: Subir cambios de código y realizar análisis de calidad de código

1. Descripción Breve

El caso de uso le permite al ingeniero desarrollador de software subir los cambios del proyecto al repositorio Bitbucket.

2. Flujo Básico de Eventos

El caso de uso inicia cuando un desarrollador de software desea subir los cambios que ha realizado en el proyecto al repositorio Bitbucket, para que así se desenlace un análisis de calidad de código.

2.1 El ingeniero desarrollador de software hace cambios a las líneas de código del proyecto y ejecuta los siguientes comandos para lograr subirlos al repositorio.

git add files

git commit –m “mensaje”

git push origin master

2.2 El caso de uso termina.

3. Flujos Alternativos

3.1 Cancelar, el desarrollador de software cierra el terminal.

4. Escenarios Claves

5. Precondiciones

5.1 Debe haber cambios en el proyecto.

5.2 El código fuente debe estar enlazado a un repositorio Bitbucket.

6. Post-Condiciones

6.1 Si el caso de uso finaliza correctamente se suben todos los cambios al repositorio Bitbucket.

7. **Puntos de Extensión**

8. **Requerimientos Especiales**

El caso de uso no contiene requerimientos especiales.

9. Información Adicional

9.1 **Prototipo Visual**

9.2 **Información Complementaria**

9.3 **Reglas de Negocio Aplicables**

Anexo 13 Caso de uso Realizar despliegue automatizado

Especificación del Caso de Uso: Realizar despliegue automatizado

1. Descripción Breve

El caso de uso le permite al ingeniero de infraestructura realizar un despliegue a partir de un build.

2. Flujo Básico de Eventos

2.1 El caso de uso inicia cuando un ingeniero de infraestructura desea realizar un despliegue de alguna versión del software.

2.2 El sistema muestra la opción de crear un release.

2.3 El ingeniero de infraestructura hace clic en el botón Create release.

2.4 El ingeniero de infraestructura hace clic en .

2.5 El sistema muestra la siguiente interfaz.

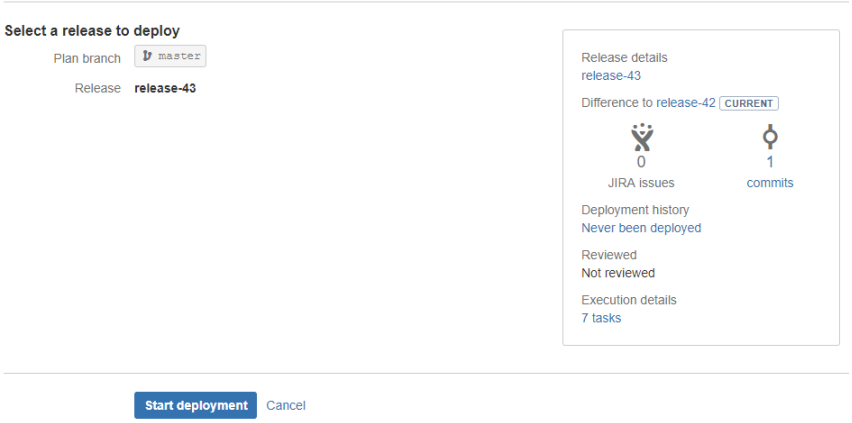


Figura 37. Iniciar despliegue

Fuente: Bamboo

2.6 El ingeniero de infraestructura hace clic en Start deployment.

2.7 Al finalizar el sistema muestra la siguiente interfaz.

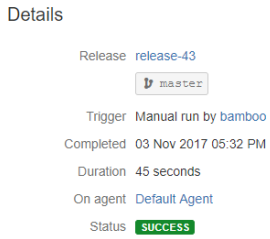


Figura 38. Resultado despliegue
Fuente: Bamboo

2.8 El ingeniero de infraestructura comprueba despliegue.

2.9 El caso de uso termina.

3. Flujos Alternativos

3.1 Página Inicio, el ingeniero de infraestructura lo selecciona para cambiar de página.

4. Escenarios Claves

5. Precondiciones

5.1 Deben existir builds creados en Bamboo.

5.2 El servidor de producción debe estar iniciado.

6. Post-Condiciones

6.1 Si el caso de uso finaliza correctamente el sistema realiza el despliegue exitosamente.

7. **Puntos de Extensión**

8. **Requerimientos Especiales**

El caso de uso no contiene requerimientos especiales.

9. Información Adicional

9.1 **Prototipo Visual**

9.2 **Información Complementaria**

Anexo 14 Caso de uso Realizar rollback automatizado

Especificación del Caso de Uso: Realizar rollback automatizado

1. Descripción Breve

El caso de uso le permite al ingeniero de infraestructura realizar un rollback de la aplicación mas no de la base de datos.

2. Flujo Básico de Eventos

2.1 El caso de uso inicia cuando un ingeniero de infraestructura desea realizar un rollback de alguna versión anterior del software.

2.2 El ingeniero de infraestructura selecciona el proyecto de despliegue haciendo clic en .

2.3 El sistema muestra la siguiente interfaz.

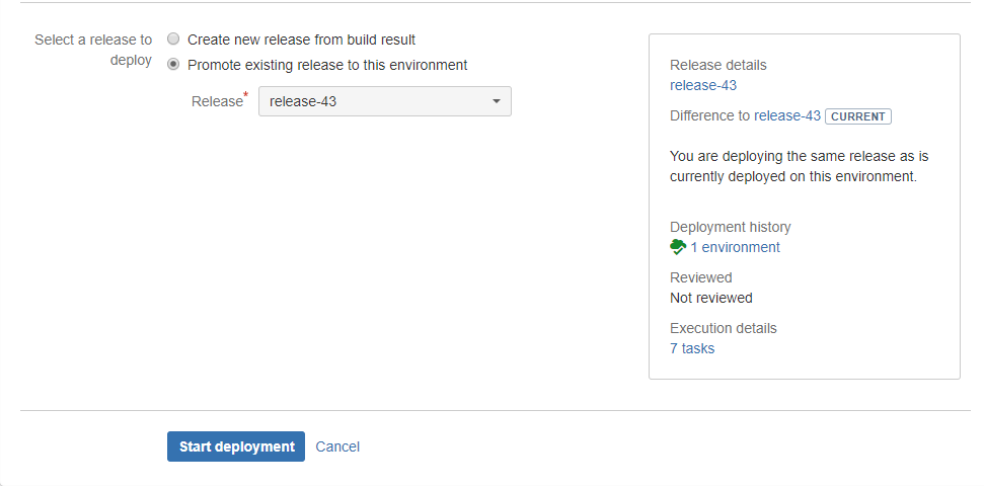


Figura 39. Interfaz despliegue
Fuente: Bamboo

2.4 El ingeniero de infraestructura selecciona el release al cual quiere hacer rollback y hace clic en Start deployment

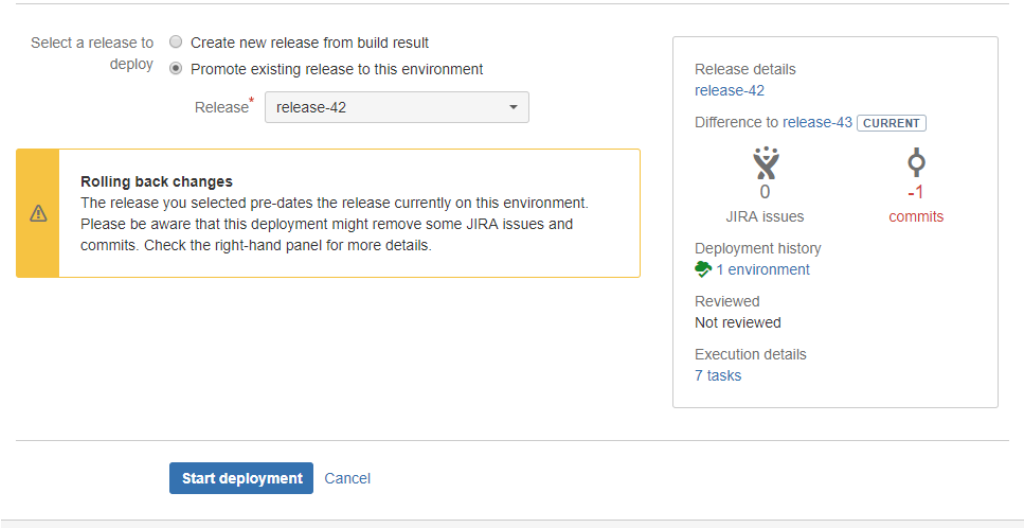


Figura 40. Iniciar rollback
Fuente: Bamboo

2.5 El ingeniero de infraestructura hace clic en Start deployment.

2.6 Luego el sistema muestra la siguiente interfaz.

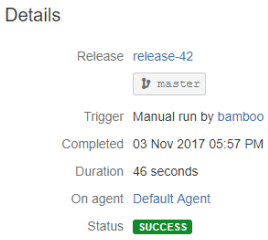


Figura 41. Resultado rollback
Fuente: Bamboo

2.7 El ingeniero de infraestructura comprueba el despliegue.

2.8 El caso de uso termina.

3. Flujos Alternativos

3.1 Página Inicio, el ingeniero de infraestructura lo selecciona para cambiar de página.

4. Escenarios Claves

5. Precondiciones

5.1 Deben existir releases previos en Bamboo.

5.2 El servidor de producción debe estar prendido.

6. Post-Condiciones

6.1 Si el caso de uso finaliza correctamente el sistema realiza el rollback exitosamente.

7. Puntos de Extensión

8. **Requerimientos Especiales**

El caso de uso no contiene requerimientos especiales.

9. Información Adicional

10. Prototipo Visual

10.1 **Información Complementaria**

10.2 **Reglas de Negocio Aplicables**

Anexo 15 Caso de uso Realizar despliegue manual

1. Conectarse con el servidor de producción.
2. Dirigirse a la siguiente ruta:

```
cd /home/ubuntu/scriptsForPreTest
```

3. Eliminar el proceso en el puerto 8000 ejecutando el siguiente script

```
. killProcessPort8000.sh
```

4. Descargar nuevos cambios del repositorio:

```
. pullCodePretest.sh
```

5. Ejecutar el siguiente comando para eliminar el directorio

```
sudo rm -r pretest
```

6. Dirigirse a la siguiente ruta:

```
cd /home/ubuntu/scriptsForPreTest
```

7. Ejecutar el siguiente comando actualizar el código fuente.

```
. RemoveCodeAndPullCode.sh
```

8. Ejecutar el siguiente comando para cambiar los permisos al directorio.

```
sudo chmod 777 -R pretest
```

9. Cambiar los permisos del siguiente archivo.

```
sudo chmod -R 777 /tmp/python.log
```

10. Ingresar nuevamente a la ruta:

```
cd /home/ubuntu/scriptsForPreTest
```

11. Ejecutar el siguiente comando para reiniciar el servicio de NGINX

```
sudo service nginx restart
```

12. Ejecutar el siguiente comando para activar el entorno virtual.

```
cd /home/ubuntu/
source env/bin/activate
```

13. Ejecutar el siguiente comando para ejecutar la aplicación.

```
nohup gunicorn project.wsgi:application --bind 0.0.0.0:8000 </dev/null>/dev/null 2>&1 &
```

Anexo 16 Descripción general del proyecto portal de empleo

El proyecto portal de empleo es una aplicación desarrollada en Python que permite publicar ofertas de trabajo, los actores se muestran a continuación:

Actores

- Empresa
- Postulante
- Administrador

Modelo de casos de uso

El modelo de casos de uso del proyecto portal de empleo es el siguiente (ver Figura 42).

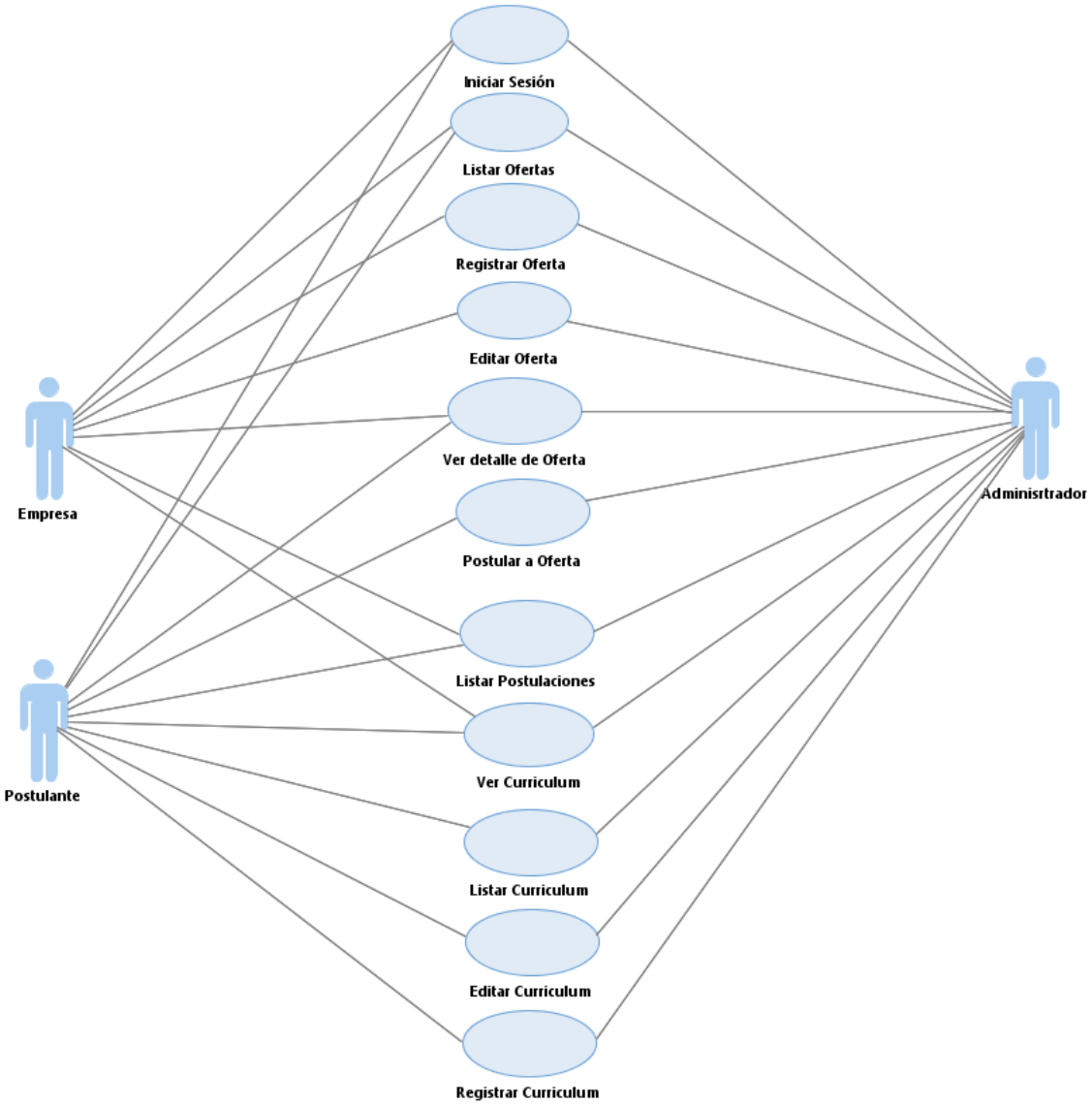


Figura 42. Casos de uso del proyecto portal de empleo

Diagramas de flujo

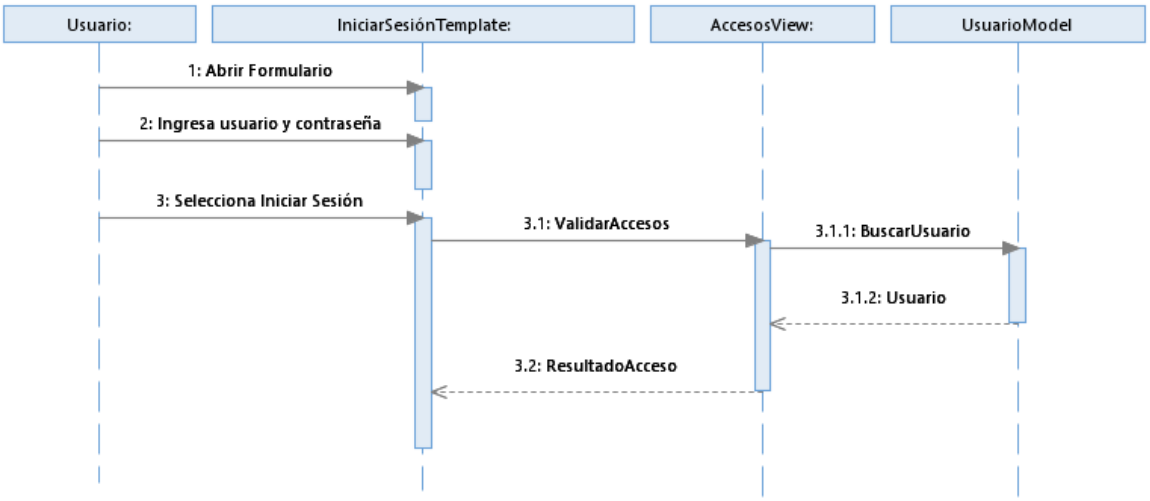


Figura 43. Diagrama de flujo iniciar sesión

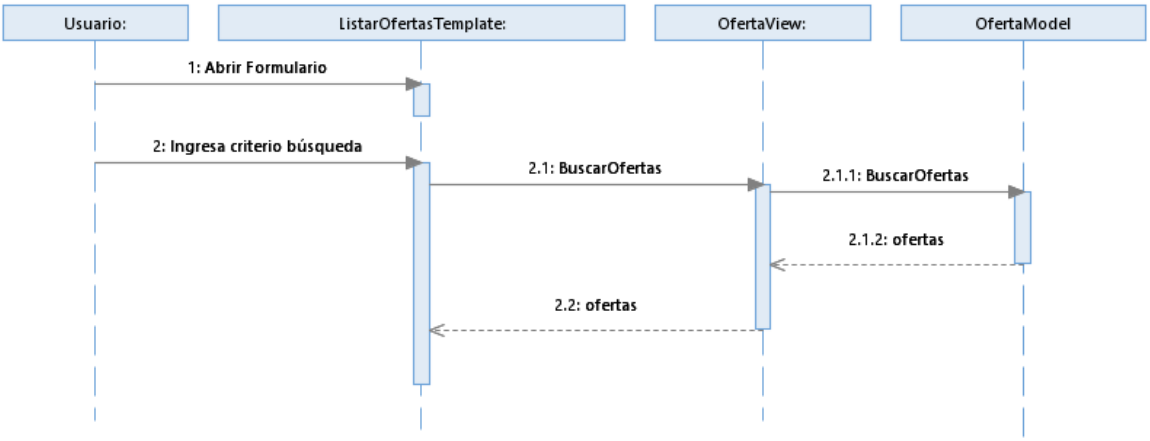


Figura 44. Diagrama de flujo listar ofertas

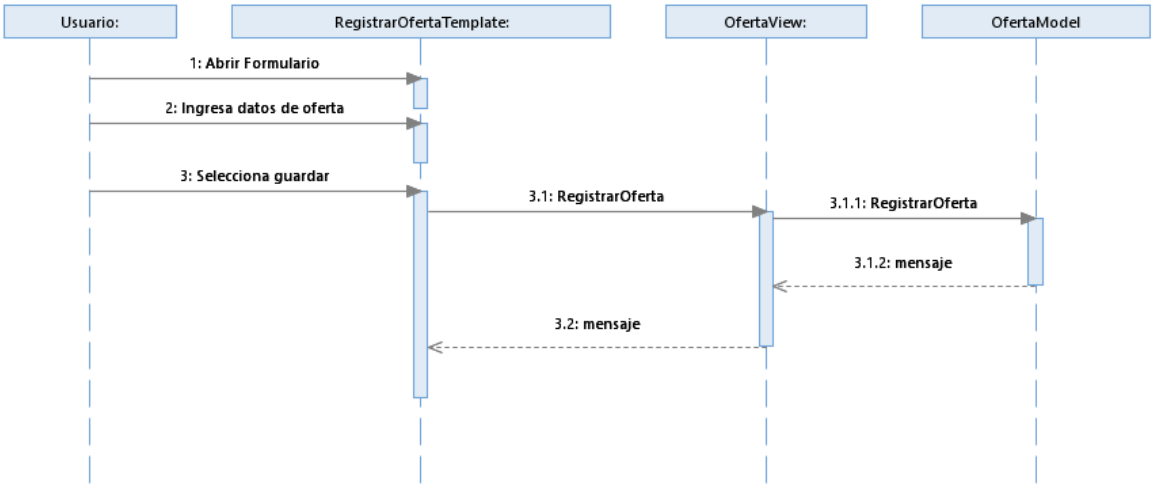


Figura 45. Diagrama de flujo registrar oferta

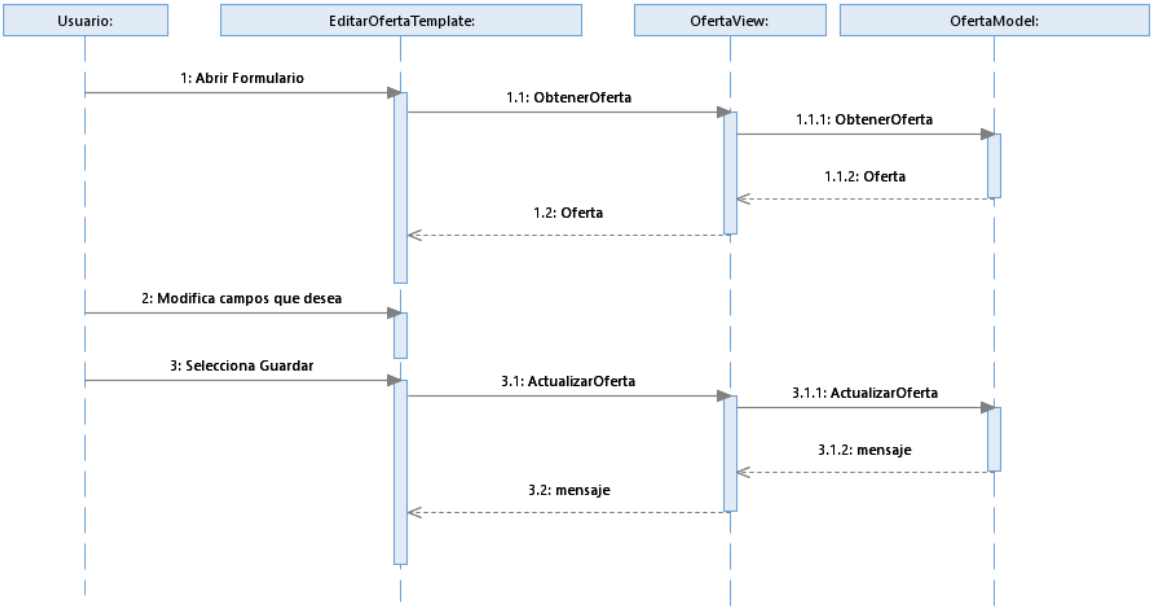


Figura 46. Diagrama de flujo editar oferta

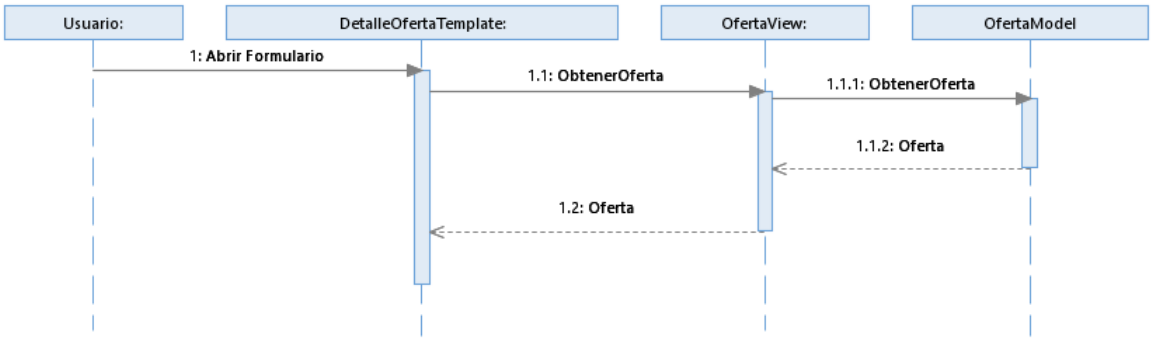


Figura 47. Diagrama de flujo ver detalle de oferta

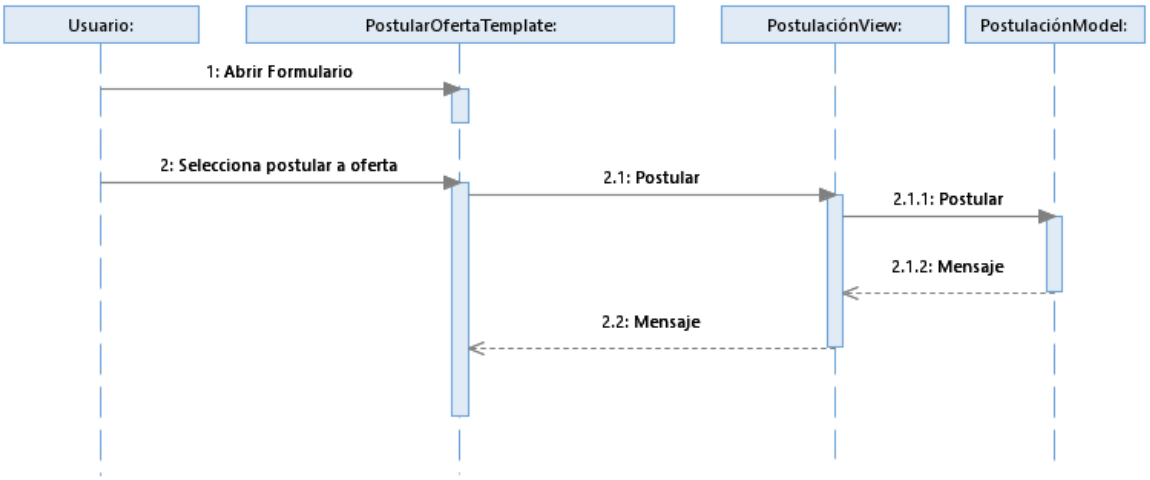


Figura 48. Diagrama de flujo postular a oferta

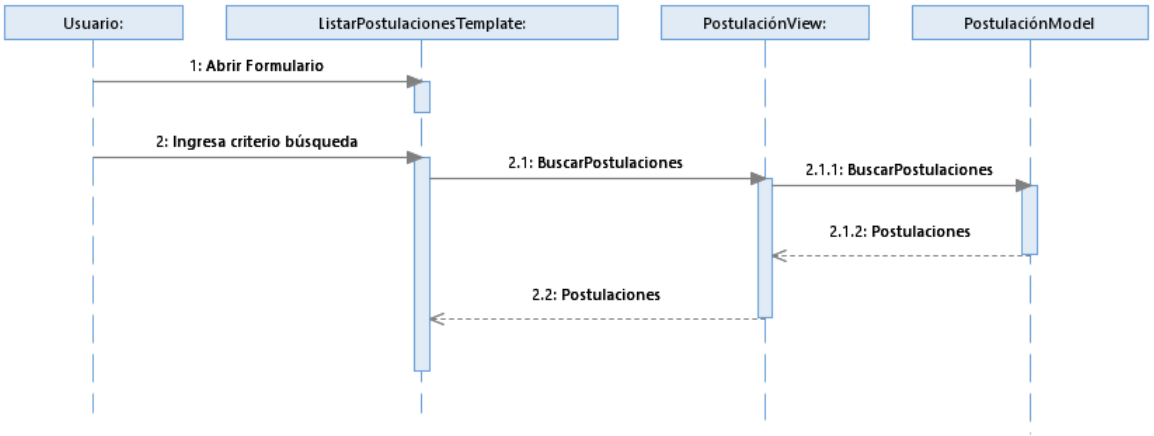


Figura 49. Diagrama de flujo listar postulaciones

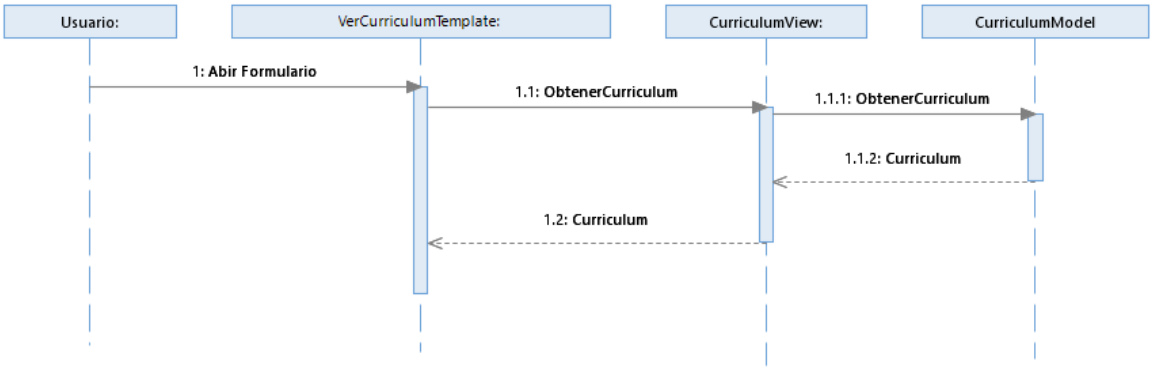


Figura 50. Diagrama de flujo ver curriculum

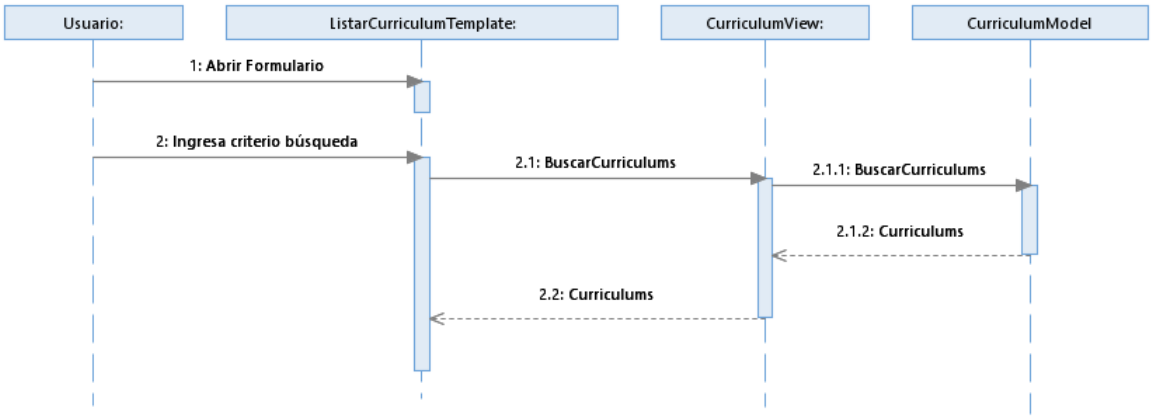


Figura 51. Diagrama de flujo listar curriculum

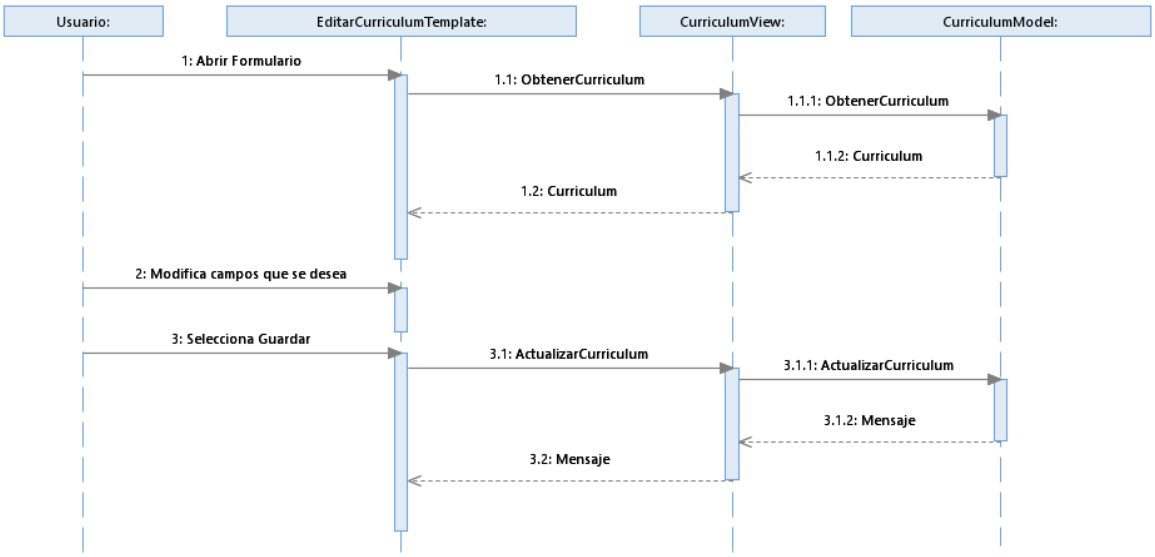


Figura 52. Diagrama de flujo editar curriculum

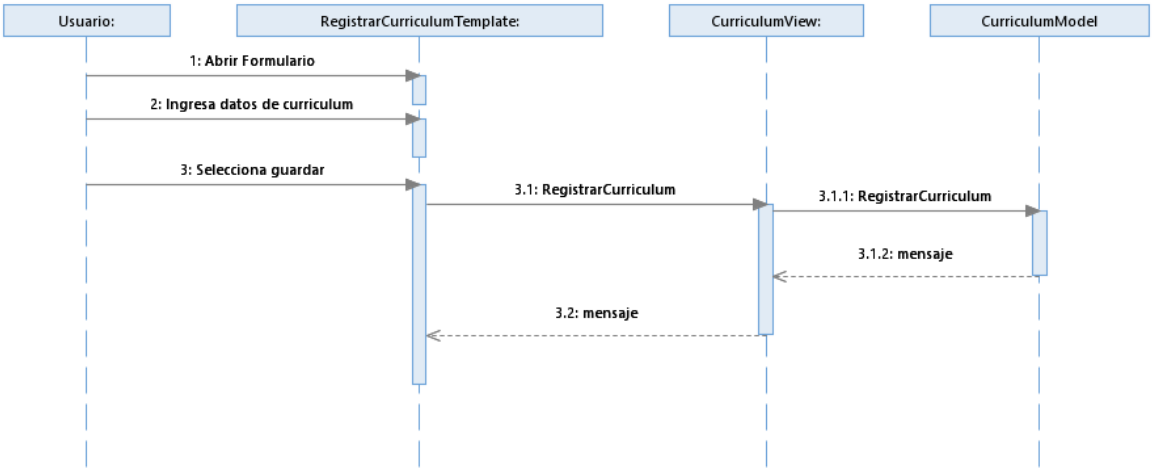


Figura 53. Diagrama de flujo registrar curriculum

Diagramas de clases

Figura 54. Diagrama de clases iniciar sesión

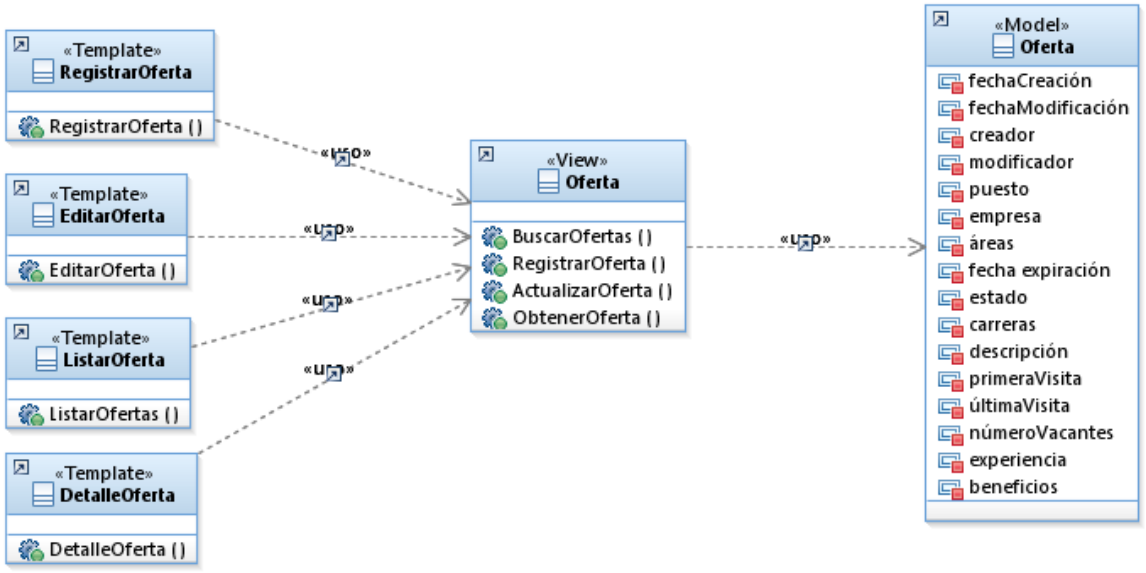


Figura 55. Diagrama de clases ofertas

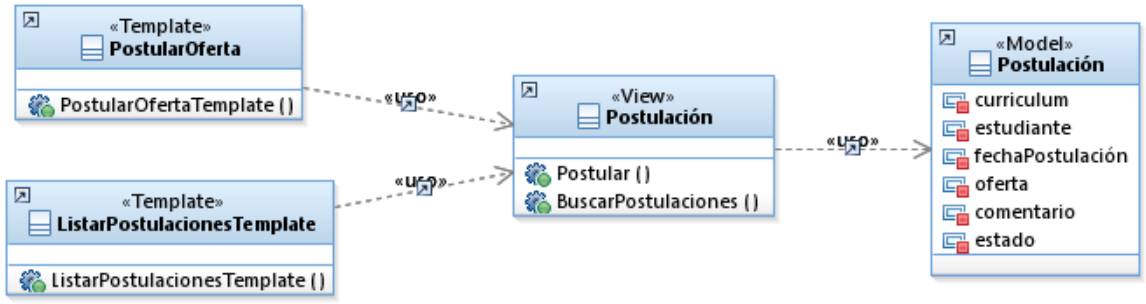


Figura 56. Diagrama de clases postulación

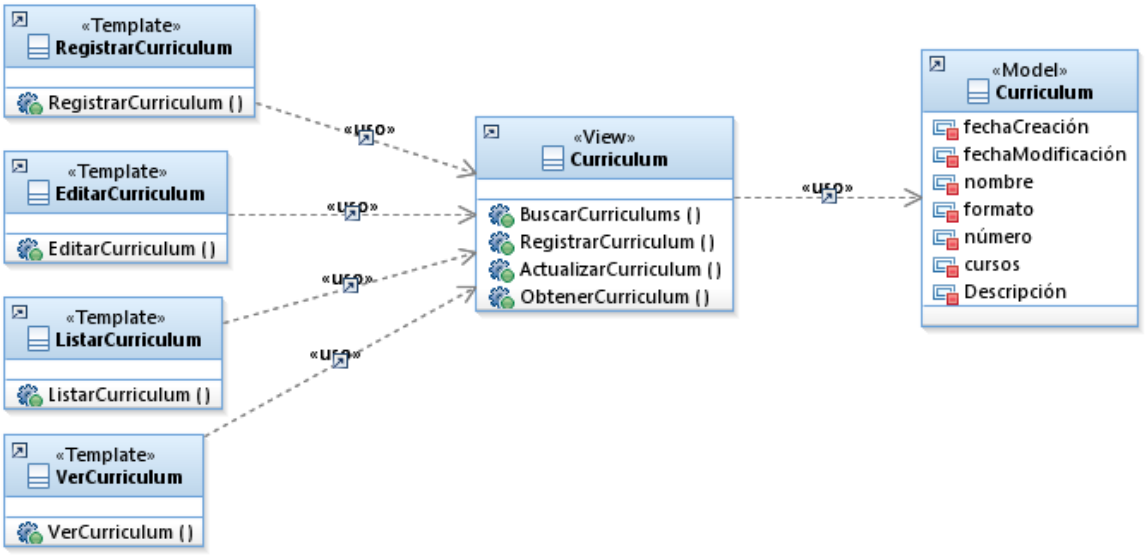


Figura 57. Diagrama de clases curriculum

Arquitectura

La arquitectura del proyecto portal de empleo se basa en el patrón MVC, la cual se muestra en la Figura 58.

Nduka Oyom (2017), define los componentes de este patrón de la siguiente manera:

Modelo: Maneja la representación de los datos, este sirve como una interfaz para los datos guardados en la base de datos, y también permite interactuar con los datos.

Vista: Representa lo que el usuario ve en el navegador web de una aplicación web.

Controlador: Provee la lógica para manejar el flujo de representación en la vista o actualizar datos de modelo de datos.

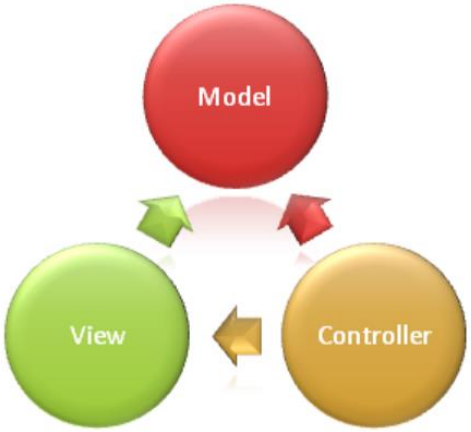


Figura 58. Arquitectura MVC
Fuente: (Nduka Oyom, 2017)

Ya que la aplicación usa el Framework Django, entonces se considera el patrón MTV (Model Template View), la cual es muy parecida al patrón MVC, en donde hay una relación la cual se muestra en la Figura 59.

Modelo: Exactamente igual al modelo del patrón MVC.

Plantilla: Cumple la misma función que la vista del patrón MVC**.**

Vista: Trabaja exactamente igual al controlador del patrón MVC.

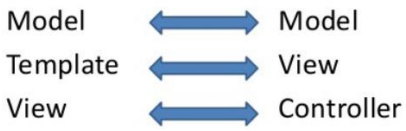


Figura 59. Comparación Arquitectura MTV y MVC
Fuente: (Nduka Oyom, 2017)

Diagrama de despliegue

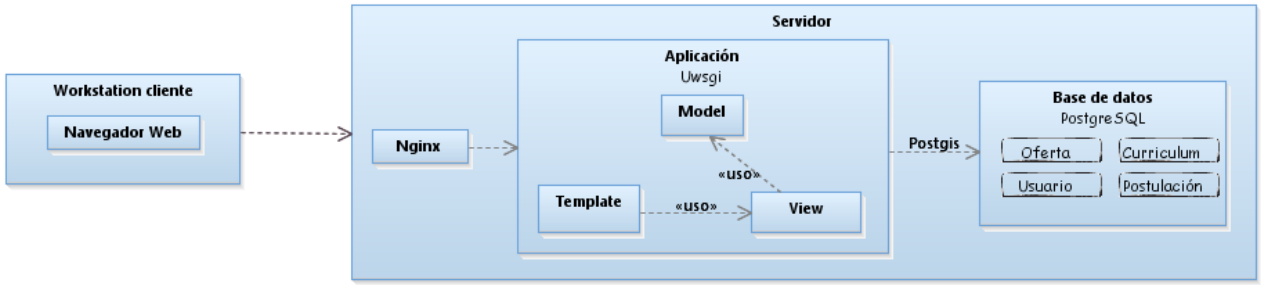


Figura 60. Diagrama de despliegue

Anexo 17 Elección de herramientas

Bitbucket

- Ofrece repositorios privados de código fuente gratuitamente.
- Presenta una interfaz gráfica amigable.
- Facilidad de integración con herramientas como, por ejemplo: Bamboo.

Bamboo:

- Soporte directo de Atlassian.
- Presenta interfaz gráfica amigable.
- Permite la automatización de tests.
- Integra tareas que permiten realizar despliegues fácilmente.

Sonarqube:

- Mide indicadores de calidad de código, teniendo como base a estándares como es CWE.
- Indica la ubicación de cada resultado de la medición con ejemplos que guían fácilmente a mejorar la calidad de código.