Creación Automática de Pipelines Para CI/CD Usando Técnicas de Reutilización y Variabilidad

Robinson Cruz Delgado

Anteproyecto presentada(o) como requisito parcial para optar al título de:
Magister en Ingeniería de Software

Director(a):
Ph.D. Jaime Chavarriaga

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería
Departamento de Electrónica y Ciencias de la Computación
Cali, Colombia
20 de julio de 2022

Ficha Resumen
Anteproyecto de Trabajo de Grado

Posible Título: Creación Automática de Pipelines Para CI/CD Usando Técnicas de Reutilización y Variabilidad

1. **Área de trabajo:** CI/CD y Líneas de Productos de Software
2. **Tipo de proyecto (Aplicado, Innovación, Investigación):** Aplicado
3. **Estudiante:** Robinson Cruz Delgado
4. **Correo electrónico:** robcruzd@javerianacali.edu.co
5. **Dirección y teléfono:** Cra 19f # 27 - 33 Sur Olaya Bogotá, 3135818403
6. **Director:** Jaime Chavarriaga
7. **Vinculación del director:** Aranda Software
8. **Correo electrónico del director:** jaime.chavarriaga@gmail.com
9. **Co-Director:** Luisa Fernanda Rincón Pérez
10. **Grupo o empresa que lo avala (Si aplica):**
11. **Otros grupos o empresas:**
12. **Palabras clave(al menos 5):** CI/CD, Continuous Integration, Continuous Delivery, Continuous Deployment, Devops, Pipeline, Automation, Reuse, Variability, Software Product Lines
13. **Fecha de inicio:** 1 de Agosto de 2022
14. **Duración estimada (en meses):** 6 meses

15. **Resumen:** La integración y despliegue continuo permiten a las empresas entregar constantemente nuevas funcionalidades de software a sus usuarios. Algunas empresas despliegan varias veces por minuto o por día sin ningún inconveniente. Para lograrlo, los desarrolladores implementan uno o más pipelines que ejecutan tareas como descargar código fuente, ejecutar pruebas, construir el software, generar paquetes e instaladores y desplegar la aplicación. Estos pipelines, que ahora son parte del código fuente, han terminado incorporando sus mismos problemas relacionados a malas prácticas de desarrollo y de configuración, introduciendo vulnerabilidades al proceso, problemas de confiabilidad y sobrecostos durante su ejecución. Debido a esto, algunos autores han propuesto soluciones para generar pipelines automáticamente mediante especificaciones de alto nivel o modelos de arquitectura. Sin embargo, estos esquemas generan pipelines, uno por cada ambiente y con tecnologías definidas, sin reutilizar código o aprovechar técnicas de manejo de variabilidad. Por ello, se plantea la creación de una herramienta que aproveche las cualidades de la reutilización y la variabilidad de software, generando automáticamente pipelines para CI/CD y mitigando los problemas ya mencionados. El dar solución a esta problemática, puede ser el punto de partida para crear una herramienta más robusta y con mayor variabilidad para crear automáticamente pipelines en CI/CD y así resolver los problemas relacionados a ellos.

Índice

Índice de figuras

Resumen

La integración y despliegue continuo, junto con los pipelines como código, han generado una revolución en la industria del software permitiendo publicar cambios desde el código en menos de una hora en producción. Sin embargo, los pipelines como código agregan los mismos problemas del código en general, como malas practicas de desarrollo y configuración, vulnerabilidades, desconfianza o sobrecostos durante su ejecución. Aunque, autores han creado herramientas para generar automática pipelines e intentar solventar estos problemas, ninguno aprovecha técnicas de variabilidad y reutilización de software. Por ello, se plantea la creación de una herramienta que aproveche las cualidades de estas técnicas, para crear automáticamente pipelines de CI/CD mitigando los problemas ya mencionados. Al dar solución a esta problemática, puede ser el punto de partida para crear una herramienta más robusta y con mayor variabilidad para crear automáticamente pipelines en CI/CD y así resolver los problemas relacionados a ellos.

Palabras Clave Integración continua, despliegue continuo, pipeline, reutilización de software, variabilidad de software

Abstract

Continuous integration and ontinuous deployment, along with pipelines as code, have sparked a revolution in the software industry by allowing changes to be released from code in less than an hour to production. However, pipelines as code add the same problems as code in general, such as poor development and configuration practices, vulnerabilities, mistrust or cost overruns during execution. Although authors have created tools to automatically generate pipelines and try to solve these problems, none take advantage of software variability and reuse techniques. For this reason, the creation of a tool that takes advantage of the qualities of these techniques is proposed, to automatically create CI/CD pipelines, mitigating the aforementioned problems. By solving this problem, it can be the starting point to create a more

robust tool with greater variability to automatically create pipelines in CI/CD and thus solve the problems related to them.

Keywords CI/CD, pipeline, software reuse, software variability

1. Introducción

El gran auge de la industria del software en los últimos años, en los cuales muchas empresas han realizado su transición a ese mundo digital, ha ampliado la competencia y en el mismo sentido la necesidad de ser el primero en brindar al publico nuevos servicios y características que incentiven a los clientes a ir hacia ellos o mantenerse. Con esa idea de brindar un camino o mejor aún, pavimentar el ya existente para llevar esos nuevos desarrollos hasta producción de una forma más ágil, con respecto a las pruebas y despliegues manuales tradicionales, se incorpora en la industria la Integración Continua y Entrega y Despliegue Continuo (CI/CD).

En estos procesos de Integración, Entrega y Despliegue, se involucran muchas tareas que tienen como objetivo validar el correcto funcionamiento del nuevo desarrollo a publicar; como lo son, pruebas unitarias, pruebas de integración, pruebas funcionales, construcción de los artefactos, entre otros, que procuran validar ese cambio. Para lograr las velocidades de despliegue que tienen algunas empresas de élite como Amazon, quienes despliegan nuevas características varias veces por segundo, se tienen los pipelines como código, los cuales ejecutan cada uno de los pasos mencionados previamente y van llevando ese código nuevo hasta producción de una forma automática. Sin embargo, aunque todo el proceso en los pipelines puede ser automático, la generación de los pipelines se realiza de forma manual y con tácticas de reutilización oportunista, como lo son copiar y pegar de archivos ya existentes.

El presente trabajo de grado, busca crear una herramienta que permita generar automáticamente pipelines para CI/CD, usando técnicas de reutilización y variabilidad de software. Acotando la herramienta a arquitecturas de aplicaciones web y los lenguajes, frameworks y mecanismos más usados en ellas en la actualidad. Dando un primer paso en la automatización de pipelines mediante estas técnicas.

Para desarrollar esta herramienta de generación de pipelines, el primer paso en esta propuesta es la identificación de las técnicas adecuadas de variabilidad y reutilización que se adaptan al problema. Además, la definición de las herramientas que tendrán las arquitecturas de los proyectos a los cuales se les va a generar los pipelines. Para así, diseñar la herramienta de generación automática bajo estos criterios e implementarla.

Cuando finalmente se tenga la herramienta funcionando, se propone hacer una validación inicial con proyectos de prueba a los cuales se les genere el pipeline de despliegue y se verifique su correcto funcionamiento. Al obtener este objetivo exitoso, se procederá a validar con expertos en creación de pipelines para CI/CD, con quienes se realizará la prueba en un ambiente más real y se consultará con ellos su opinión respecto a utilidad y satisfacción con la herramienta.

2. Definición del problema

2.1. Planteamiento del problema

En la actualidad la industria de software tiene una gran presión para entregar valor más rapidamente, introduciendo nuevas funcionalidades con mayor velocidad y sin afectar la continuidad de los servicios que ofrecen a sus clientes. Por ejemplo, empresas como Microsoft, Google y Amazon liberan constantemente nuevas funcionalidades de sus productos, varias veces al día, e incluso, varias veces por segundo [1].

Según un estudio del *DevOps Research and Assessment*, el tiempo de espera para cambios en el software en empresas de élite se ha reducido de aproximadamente un día en 2019, a menos de una hora en 2021. Estas empresas de élite logran un tiempo de entrega 6570 veces más rapido, desde que se realiza un commit en un repositorio de software, que las empresas que tienen un bajo nivel de implementación de prácticas de DevOps (DORA, 2021). Este tipo de mejoras en productividad solo son posibles

[1] `https://www.allthingsdistributed.com/2014/11/apollo-amazon-deployment-engine.html`

a través de herramientas de automatización que permitan hacer integración, entrega y despliegue de forma continua.

Los desarrolladores implementan estos procesos de integración, entrega y despliegue continuos (CI/CD) usando uno o más *pipelines* por cada proyecto. Estos pipelines establecen las tareas que deben ejecutarse para realizar cosas como descargar el código fuente de los repositorios de código, ejecutar pruebas, construir el software, generar paquetes e instaladores y desplegar la aplicación en ambientes de pruebas y de producción (TechTarget, 2021).

Las empresas pueden utilizar diversas herramientas para implementar CI/CD. Herramientas como Jenkins [2], Github Actions [3], Azure Pipelines [4], Travis CI [5] o Circle CI [6] pueden ser utilizadas para ejecutar los pipelines correspondientes. En estas herramientas los pipelines se especifican como parte del código fuente y se modifican y versionan como el resto del software, llamados *pipeline as code* (Labouardy, 2021). Además, estas herramientas permiten definir funciones y librerías que se pueden reutilizar en los pipelines, permiten ejecutar otros pipelines, soportan múltiples configuraciones y permiten definir combinaciones de parámetros. Sin embargo, algunos estudios muestran que los desarrolladores no aprovechan muchas de estas opciones al momento de crear los pipelines (Kinsman et al., 2021)(Chen et al., 2021)(Val, 2022).

Existen además problemas en la creación de pipelines. Por ejemplo, luego de analizar miles de pipelines en Github Actions (Kinsman et al., 2021)(Chen et al., 2021) y Travis CI (Gallaba and McIntosh, 2020): Algunos pipelines analizados se construyen siguiendo malas prácticas de desarrollo (Zampetti et al., 2020), de configuración (Vassallo et al., 2020), introducen vulnerabilidades en el proceso (Paule et al., 2019), no se ejecutan de forma confiable (Gallaba, 2021) o generan sobrecostos durante su ejecución (Jin, 2022).

Para evitar estos problemas, algunos autores han propuesto esquemas para gene-

[2] `https://www.jenkins.io/`
[3] `https://github.com/features/actions`
[4] `https://azure.microsoft.com/es-es/services/devops/pipelines/`
[5] `https://travis-ci.com/`
[6] `https://circleci.com/`

rar los pipelines a partir de especificaciones de alto nivel (Jones, 2018) o a partir de modelos de arquitectura (Aydin et al., 2021). Sin embargo, estos esquemas no consideran aprovechar técnicas de reutilización y manejo de variabilidad para la creación de varios pipelines para uno o más proyectos de software. Por ejemplo, no consideran la posibilidad de definir ambientes de desarrollo y de pruebas [7] donde se pueden reutilizar tareas del pipeline o considerar variaciones sobre estos mismos ambientes que puedan afectar los pasos de los pipelines. Estos esquemas generan pipelines, uno por cada ambiente, sin reutilizar código o aprovechar técnicas de manejo de variabilidad.

2.2. Formulación del problema

Teniendo en cuenta la necesidad de mejora en la reutilización de código en los pipelines y el manejo de su variabilidad, respecto a las herramientas que se utilizan en los proyectos de software y los diferentes ambientes en que se despliegan, se plantean los siguientes interrogantes.

- ¿Qué oportunidades de reutilización y manejo de variabilidad se pueden aplicar en la construcción de pipelines de CI/CD?
- ¿Cómo se puede implementar la generación automática de pipelines de CI/CD considerando las técnicas de reutilización y manejo de variabilidad identificadas previamente?
- ¿Qué ventajas y desventajas perciben los desarrolladores al utilizar los esquemas de generación automática que se propone en el presente proyecto?

[7] `https://launchdarkly.com/blog/test-environments-101-definition-types-and-best/`

3. Objetivos del proyecto

3.1. Objetivo General

Proponer un sotware que genere automáticamente pipelines para CI/CD variando las herramientas a usar en un proyecto y el ambiente en qué se despliega, utilizando técnicas de reutilización y variabilidad.

3.2. Objetivos específicos

- Identificar qué técnicas se han usado para mejorar la reutilización y la variabilidad de los pipelines para CI/CD y definir la que se adapte mejor al problema a resolver.
- Diseñar un software que genere automáticamente pipelines para CI/CD, mediante técnicas de reutilización y variabilidad.
- Implementar la solución diseñada, generando con ella automáticamente pipelines para CI/CD, mediante técnicas de reutilización y variabilidad.
- Demostrar el funcionamiento de la implementación usando proyectos de prueba como casos de estudio.
- Evaluar la satisfacción y utilidad de la solución planteada con personas que en su labor utilicen pipelines para CI/CD.

3.3. Resultados esperados

Mediante el presente trabajo de grado se espera obtener una herramienta que automatice la generación de pipelines para CI/CD, usando técnicas de reutilización y variabilidad de software, enfocado en algunas de las herramientas más comunes en el desarrollo de aplicaciones web. Todo lo anterior, enmarcado en una metodología de investigación formal, como lo es Design Science Research.

4. Alcance

Este trabajo de grado tiene como objetivo encontrar una nueva alternativa para la creación de pipelines para CI/CD que tienen las soluciones software. Teniendo como variables los ambientes en que estos se despliegan y la variedad de herramientas que se puede usar al crear estos proyectos de software en general. Para ello, se plantea el uso de técnicas de reúso y variabilidad en ambientes como desarrollo, pruebas y producción [8] y en relación a las herramientas, se busca usar las herramientas más comunes para el desarrollo de páginas web.

La implementación se creará apoyándose de proyectos basícos de páginas web con las herramientas que posteriormente se seleccione, con los cuales se hará el primer paso de validación, para posteriormente validarlos usando proyectos de una empresa real y de esa manera validar con los ingenieros encargados de la creación y mantenimiento de los pipelines de CI/CD la satisfacción y utilidad de la solución.

[8] `https://sdlc.uconn.edu/getting-started/`

5. Justificación del trabajo de grado

La automatización de los procesos de integración, entrega y despliegue continuos en los pipelines de los proyectos de software, ha generado una gran velocidad para publicar en el mercado cambios y nuevas versiones, pero al tiempo vinculando una gran responsabilidad en la persona encargada de la gestión de estos pipelines, ya que, estos se convierten en la base para llevar el software desde el código hasta producción.

Adicionalmente, los pipelines introducen otro problema que afecta a estas personas encargadas de su desarrollo, ya que, al ser los pipelines otra pieza de código, conlleva los mismos problemas que las líneas de código de software tienes en general; como lo son, malas prácticas, errores de configuración y vulnerabilidades en el software. Estos problemas anteriores, se pueden resolver automatizando la generación de estos pipelines, buscando la agilidad en su proceso de creación y los errores que puedan incorporar. Y aunque ya se han realizado proyectos previos que generan pipelines automáticamente, no se ha aprovechado en ellos técnicas de reutilización y variabilidad, lo cual le podría brindar otro porcentaje de eficacia a la generación y por tanto, mayor satisfacción y utilidad a la persona responsable de la gestión de pipelines y en el mismo camino al proyecto mismo por la eliminación de esos posibles problemas.

6. Marco teórico de referencia y antecedentes

6.1. Antecedentes

6.1.1. Continuous Integration (CI)

Durante mucho tiempo, los desarrolladores normalmente construían software creando módulos de software que se integraban y se probaban al final del proceso de desarrollo. En estos casos, muchos errores ya son detectados cerca de la fecha de entrega, causando reprocesos, retrasos en el proyecto y frustración entre los desarrolladores (Duvall et al., 2007). Este fenómeno conocido como "integration hell" o "merge hell" resultaba ser un problema considerable a medida que los productos de software se volvían más grandes y complejos.

La **Integración Continua (CI)** es una práctica de software donde los diferentes módulos de software se integran y se prueban frecuentemente durante el transcurso del proyecto [9]. El término fue propuesto inicialmente por Grady Booch para explicar como se integran las versiones internas cuando se realiza desarrollo de forma iterativa (Booch, 1990). La práctica fue popularizada posteriormente por Kent Beck y Ron Jeffries como parte de las técnicas de *Extremme Programming (XP)* (Beck, 1999). Posteriormente, otras metodologías ágiles, como Lean Software Development (Poppendieck and Poppendieck, 2003), también incluyeron CI como una de sus prácticas; grandes empresas, como Ericsson (Karlsson et al., 2000) y Microsoft (Cusumano and Selby, 1998), empezaron a usarlo como parte de su desarrollo y una variedad de herramientas surgieron para soportar su realización.

Entre las primeras herramientas que soportaban CI, podemos mencionar: CruiseControl (2001) [10], Hudson (2008) [11] y Jenkins (2011)[12], que es una versión derivada que surgió luego de la adquisición de Hudson por parte de Oracle (2011). En la actualidad, muchos desarrolladores han reemplazado estas herramientas por servicios de cloud computing que soportan integración continua (Gol, 2021). Entre estos servicios

[9] `https://martinfowler.com/articles/continuousIntegration.html`
[10] `https://web.archive.org/web/20180613010803/http://cruisecontrolnet.org/`
[11] `https://projects.eclipse.org/projects/technology.hudson`
[12] `https://www.jenkins.io/`

se pueden mencionar: TravisCI (2011) [13], Circle CI (2014) [14], AppVeyor (2014) [15], Gitlab Ci (2015) [16] y Github Actions (2019)[17].

Las prácticas de Integración Continua fueron documentadas completamente en (Duvall et al., 2007). En este libro se plantean seis tareas como parte de CI:

1. Integración Continua de Código
2. Integración Continua de Base de datos
3. Pruebas Continuas
4. Inspección Continua
5. Entrega Continua
6. Retroalimentación Continua

Siendo la pruebas automáticas una parte integral de los CI/CD pipelines, en (Labouardy, 2021), describen los tipos de pruebas más famosas para asegurar que el software cumple con los requerimientos iniciales.

1. Pruebas unitarias
2. Pruebas de calidad
3. Pruebas de seguridad
4. Pruebas de interfaz de usuario
5. Pruebas de integración

6.1.2. Continuous Integration y Continuous Delivery (CI-CD)

En 2010, el concepto de Integración Continua fue extendido para incluir tareas adicionales relacionadas con la distribución y el despliegue del software (Humble and Farley, 2010).

[13] `http://travis-ci.org/`
[14] `http://circleci.com/`
[15] `http://www.appveyor.com/`
[16] `http://docs.gitlab.com/ee/ci`
[17] `http://github.com/features/actions`

La **Integración y Entrega Continua (CI-CD)** define un proceso de extremo a extremo que incluye el desarrollo de software, su distribución a los clientes y su despliegue en ambientes de pruebas y producción. De acuerdo a los autores (Humble and Farley, 2010), la construcción y la instalación manual del software son anti-patrones que deben evitarse en el proceso de desarrollo.

En este marco de trabajo podemos identificar tres tipos de tareas que se ejecutan continuamente y se automatizan con herramientas especializadas:

- **Integración Continua:** construcción continua del software, integrando y probando los diferentes módulos de software.
- **Entrega Continua:** generación de paquetes, imágenes e instaladores del software que pueden entregarse a los clientes.
- **Despliegue Continuo:** instalación y puesta en producción para que los usuarios puedan utilizar el software.

6.1.3. Deployment Pipelines

En (Humble and Farley, 2010), introducen el concepto de *Deployment Pipeline* como "la automatización del proceso que toma el software del sistema de control de versiones y lo entrega en las manos de los clientes". Cabe mencionar, que aunque el objetivo es automatizar todo el proceso, es posible tener pasos manuales, y de igual manera se tiene un pipeline definido.

Sin embargo, con el objetivo de lograr un nivel de automatización completo en el proceso, limitando la intervención humana y posibles retrasos en los despliegues, surgieron los *pipelines as code* (Labouardy, 2021), los cuales hacen parte del código de los proyectos y se modifican y versionan como el resto del software. De esta manera, las herramientas anteriormente mencionadas en integración continua, permiten el definir funciones y librerías que se pueden reutilizar en los pipelines, además de ejecutar otros pipelines, soportar múltiples configuraciones y permitir definir combinaciones de parámetros.

En (Lachhman, 2021) definen algunas consideraciones importantes a tener en cuenta en el diseño de pipelines:

- Calidad y cobertura/automatización de pruebas
- Seguridad
- Configuración del entorno de destino
- Variables y secretos
- Estrategia de lanzamiento (azul/verde/*canary*)
- Aprobaciones
- Auditoría y cumplimiento
- Estrategias de reversión y fallas
- Mantenimiento de ANSs, ONSs e INSs

6.1.4. Técnicas de Reutilización y variabilidad

Las técnicas de reutilización de software, tiene diferentes caras dependiendo de la técnica a utilizar, por ejemplo la reutilización oportunista, según (Mäkitalo et al., 2020), consiste en la creación de nuevos programas utilizando fragmentos de código ya construidos en proyectos anteriores y que no fueron desarrollados especialmente para ser reutilizados. Y por tanto, al ser códigos caóticos o con problemas, aumentan los problemas técnicos de administración, mantenimiento y evolución del código en general, como lo describe (Assunção et al., 2018).

En el lado contrario existen las técnicas de reutilización de software basados en patrones, las cuales vienen desde los años 90, donde se definieron en, (Gamma et al., 1994), algunos de los patrones más populares y comunes de hoy en día. Para ello, definen soluciones genéricas a problemas conocidos, apalancados de técnicas de reutilización de software.

Basados en los dos anteriores ejemplos, se observa el potencial que tienen las técnicas de reutilización, pero que deben implementarse con cuidado para sacar su mejor provecho. A continuación se presentan los tipos de reutilización como los definen en (Anaya, 1999):

- Por generación

- Por composición
- Desarrollo para reutilizar
- Desarrollo con reutilización
- Caja Blanca
- Caja Negra
- Adaptativo
- Interno
- Externo
- Vertical
- Horizontal
- Según el estado del proceso de desarrollo
- Según el nivel de abstracción
- Según la naturaleza del conocimiento

En relación a la variabilidad, en (Bosch and Svahnberg, 2002), la definen como la habilidad a estar sujeto a variación. Por tanto, esta técnica se enfoca en definir los dominios de variedad que existen en los componentes a usar en algunas situaciones. Como por ejemplo, en (Renault, 2014), describen algunas variabilidades en relación a los automóviles, como lo son, la diversidad de demanda de los clientes respecto a motor, tamaño de cabina, color, entre otros. O también las regulaciones que tiene cada país para los autos que circulan en ellos o el clima o infraestructura que estos tienen, lo cual genera que se deban tener consideraciones especiales para cada uno de estos dominios y así sacar el mayor provecho de cada uno de ellos.

6.2. Estado del Arte

En la revisión de estudios previamente realizados que se relacionen con el objetivo de crear pipelines automáticamente, se encontraron los siguientes trabajos.

En el artículo titulado "*Using Code Generation to Enforce Uniformity in Software Delivery Pipelines*", por (Jones, 2018), describe la creación de un generador de pipelines para Jenkins, basado en plantillas y lenguajes específicos de dominio (DSL), el cual lo llama Software Pipeline as a Service (SPaaS). El cual surgió ante la necesidad de migrar 300 aplicaciones a AWS y por tanto, querían hacer menos traumática la transición tomando los múltiples pipelines que ya tenían funcionando y creando un generador con las técnicas previamente mencionadas. Sin embargo, mencionan la dificultad de ampliar su funcionamiento a otras tecnologías, debido al DSL ya definido. Lo cual se podría permitir con técnicas de reutilización y variabilidad.

Adicionalmente, en el artículo "*Automated Construction of Continuous Delivery Pipelines from Architecture Models*", por (Aydin et al., 2021), describen la construcción automática de pipelines para entrega continua basados en el conocimiento de arquitectura de software y una vista centrada en artefactos. Buscando cerrar la brecha de conocimiento que se debe tener al crear un modelo del proceso de entrega en un software definido. Para ello, lo probaron con un pequeño grupo de 8 desarrolladores, con lo cual se noto que la mitad del grupo que eran desarrolladores Junior y tenía menos experiencia sobre arquitecturas, logró crear los artefactos necesarios, aunque para el resto del grupo con más experiencia, no fue tan intuitivo. Por lo cual se percibe, que la solución no fue tan satisfactoria, al no ser tan útil para todo el grupo de desarrolladores y en si, para le grupo que tiene más experiencia en el tema. Adicionalmente, no se perciben temas de variabilidad en la solución, con el objetivo de obtener pipelines desde diferentes herramientas.

Por otra parte, debido a que no se encontraron otros trabajos relacionados que hubiesen procurado crear un generador automático de pipelines, a continuación, se presentan algunos artículos en que se describen procesos de creación de pipelines desde diferentes enfoques.

Primeramente, en el artículo, "*Comparing DevOps procedures from the context of a systems engineer*", por (Priyadarsini et al., 2020), presentan dos arquitecturas Devops, una montada con Azure Devops y otra con AWS. Las cuales, se comparan usando los tiempos de construcción, de procesamientos y de fallas. Lo anterior, utilizando unos proyectos ya funcionando con estas tecnologías en la empresa Valamis.

Sin embargo, este proyecto se enfoca más en la comparación entre las arquitecturas, que en la generación de diversos pipelines dependiendo la tecnología utilizada.

Otro de los artículos llamado "*Fast Delivery, Continuously Build, Testing and Deployment with DevOps Pipeline Techniques on Cloud*", por (Jumani et al., 2020), presenta la implementación de pipelines para Devops en una organización, sobre la nube de Azure; centrándose en los requerimientos de un proyecto definido. Como resultado describen la solución a grandes desafíos culturales y de infraestructura que se debieron enfrentar, teniendo en cuenta, que en este último la solución se centró en la infraestructura como código que provee la nube, lo cual genera una visión positiva al uso de infraestructuras como código para la construcción de estas soluciones.

Adicionalmente, en el artículo "*Implementation of a DevOps Pipeline for Serverless Applications*", de (Ivanov and Smolander, 2018), presenta la implementación de un pipeline automatizado para Devops, para integración continua, entrega continua y prácticas de monitoreo. Todo lo anterior mediante un enfoque serverless, lo cual generó que 18 de las 27 prácticas implementadas fueran influenciadas por las características específicas de Serverless del proyecto. Con lo cual se reafirma las ventajas de utilizar infraestructura como código para dar solución a esta tarea.

Finalmente, el último artículo llamado "*Building Lean Continuous Integration and Delivery Pipelines by Applying DevOps Principles: A Case Study at Varidesk*", en (Debroy et al., 2018), presenta la implementación de devops en una de las aplicaciones web de la empresa Varidesk. En este caso, mencionan dos desafíos a los que se enfrentaron, siendo el primero los largos tiempos de espera para que las compilaciones/lanzamientos se pongan en cola y se completen y el segundo, la falta de soporte en algunas herramientas, desde una perspectiva de nubes. Resolviendo estos temas con prácticas de contenedorización, infraestructura como código y orquestación. Lo cual, nuevamente brinda ciertas recomendaciones al tipo de infraestructura y posibles herramientas a seleccionar.

7. Metodología de la investigación

La metodología a usar en el presente trabajo de grado es *Design Science Research* (DSR), en (vom Brocke et al., 2020), la cual se enfoca en generar conocimiento a través de la creación de artefactos innovadores, yendo de esa manera por la misma línea de los objetivos planteados en el proyecto en que se plantea crear automáticamente pipelines para CI/CD usando técnicas de reuso y variabilidad.

Basado en el modelo del proceso para la metodología DSR, descrito en (vom Brocke et al., 2020), se describen las siguientes actividades:

1. Identificación del problema y motivación: En esta actividad se plantea identificar el problema a resolver, la motivación, justificación de realizar el presente trabajo de grado, además de marco teórico y estado del arte sobre el tema a desarrollar.

2. Definir los objetivos para una solución: En esta actividad se definen los objetivos del presente trabajo de grado, siendo la base para resolver el problema descrito en la anterior actividad.

3. Diseño y desarrollo: En esta actividad se busca desarrollar los primeros 3 objetivos específicos del proyecto, identificando en ella las técnicas y herramientas a usar en el proyecto, para así diseñar una solución que se adapte mejor al problema a resolver, y de esa manera, finalmente realizar su implementación.

4. Demostración: En esta actividad se plantea demostrar el funcionamiento de la implementación usando proyectos de prueba como casos de estudio.

5. Evaluación: En esta actividad se evaluará la satisfacción y utilidad de la solución planteada probándola con personas que en su labor utilicen pipelines para CI/CD y obteniendo su retroalimentación mediante encuestas.

6. Comunicación: En esta actividad se generará la monografía resultante del presente trabajo de grado, además de un artículo que describa el proceso desarrollado y sus resultados obtenidos.

8. Recursos a emplear

Esta sección resume los recursos que van a emplearse en el desarrollo del proyecto de grado. Los recursos que deben considerarse incluyen a las personas que estarán involucradas en el proyecto y a los materiales como licencias, libros, etc. A continuación se listan los posibles recursos que se requieren en un proyecto de grado.

1. Recursos Humanos

   a) Director:PhD. Jaime Chavarriaga Se desempeña como director de desarrollo de software de la empresa Aranda Software, además es profesor de la materia de "Líneas de productos de software" en la Maestría de Ingeniería de Software de la Pontificia Universidad Javeriana sede Cali. Cuenta con un Doctorado en Ciencias de la Computación en Vrije Universiteit Brussel, otro Doctorado en Desarrollo de Software de la Universiadd de los Andes, una Maestría en Ingeniería de Software de la Universidad Polítecnica de Madrid, una Especialización en Gerencia Informática Organizacional en la Universidad ICESI, una Especialización en Investigación Educativa en Contextos Universitarios de la Universidad San Buenaventura Cali y un estudio de Pregrado en Ingeniería de Software de la Universidad San Buenaventura Cali. Su relación con el presente trabajo de grado se da gracias a su gran conocimiento en la cultura DevOps y las técnicas de integración y entrega continua, al igual que su conocimiento en líneas de productos de software.

   b) Co-Director:PhD. Luisa Fernanda Rincón Pérez Se desempeña como directora y profesora de los programas de posgrados de Ingeniería de Software de la Pontificia Universidad Javeriana sede Cali. Cuenta con un Doctorado de la Universidad París 1 Panteón-Sorbona, una Maestría en Ingeniería de Software de la Universidad Nacional de Colombia y un estudio de Pregrado en Administración de Sistemas Informáticos de la Universidad Nacional de Colombia. La relación con el presente trabajo de grado se presenta en su gran conocimiento en el desarrollo de proyectos de investigación y su

estrecha relación con temas relacionados a técnicas de reúso y variabilidad mediante líneas de productos de software.

c) Estudiante: Robinson Cruz Delgado Estudiante de la Maestría en Ingeniería de Software de la Pontificia Universidad Javeriana Sede Cali y autor del presente trabajo de grado. Ingeniero en Electrónica y Telecomunicaciones de la Universidad del Cauca. Se desempeña como Desarrollador de aplicaciones móviles y web en la Alta Consejería de la Alcaldía Mayor de Bogotá.

2. Económicos

   a) Equipos: Se necesita un equipo de computo para el desarrollo de todo el proceso de documentación, búsqueda, diseño, implementación y evaluación de la solución planteada.

   b) Software: Se necesitan licencias para los siguientes software:

      - Microsoft Office 360: Para gestión documental, presentaciones y reuniones virtuales con la directora y codirector del trabajo de grado.
      - Overleaf: Para la creación del Anteproyecto y Monografía del Trabajo de grado.
      - Herramientas de desarrollo: Licencias para las herramientas que en el proceso del trabajo de grado se seleccionen para desarrollar la solución planteada.

   c) Material bibliográfico: Acceso a bases de datos para la obtención de los artículos y libros requeridos como base en el desarrollo del trabajo de grado

9. Cronograma de actividades

NOMBRE DEL PROYECTO	LÍDER DE PROYECTO	FECHA DE INICIO	FECHA DE FINALIZACIÓN	PROGRESO GENERAL
Creación Automática de Pipelines Para CI/CD Usando Técnicas de Reuso y Variabilidad	Robinson Cruz	01-08-22	31-01-23	0%

RESULTADO DEL PROYECTO

DECLARACIÓN SOBRE EL ALCANCE

TAREAS	INICIO	FINALIZACIÓN	DÍAS	ESTADO
1. Identificación del problema y motivavión	01-08-22	25-08-22	25	**No se ha iniciado**
1.1. Definir problema y motivación	08-08-22	10-08-22	3	**No se ha iniciado**
1.2. Crear marco teórico y estado del arte	11-08-22	25-08-22	15	**No se ha iniciado**
2. Definición de objetivos	26-08-22	28-08-22	3	**No se ha iniciado**
3. Diseño y desarrollo	29-08-22	20-11-22	84	**No se ha iniciado**
3.1. Identificaón de técnicas y herramientas	29-08-22	20-09-22	23	**No se ha iniciado**
3.2. Diseño de la solución	21-09-22	20-10-22	30	**No se ha iniciado**
3.3. Implementación de la solución	21-10-22	20-11-22	31	**No se ha iniciado**
4. Demostración	21-11-22	20-12-22	30	**No se ha iniciado**
4.1. Selección de Casos de prueba	21-11-22	05-12-22	15	**No se ha iniciado**
4.2. Evaluación mediante casos de pruebas	06-12-22	20-12-22	15	**No se ha iniciado**
5. Evaluación	21-12-22	05-01-23	16	**No se ha iniciado**
5.1. Encuestas	21-12-22	05-01-23	16	**No se ha iniciado**
6. Comunicación	06-01-23	29-01-23	24	**No se ha iniciado**
6.1 Elaboración documentos	06-01-23	29-01-23	24	**No se ha iniciado**
Presentación	30-01-23	31-01-23	2	**No se ha iniciado**

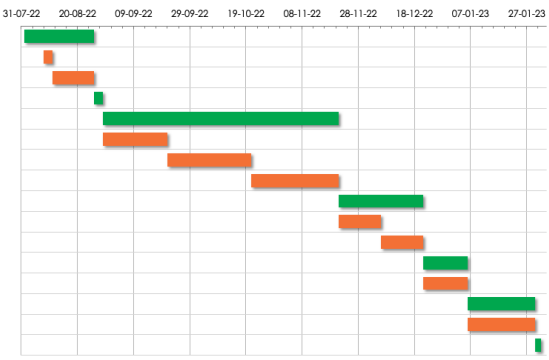

Figura 1: Cronograma de actividades

10. Referencias Bibliográficas

Referencias

(2021). *On the rise and fall of CI services in GitHub*. IEEE.

(2022). *Evolution of GitHub Action Workflows*.

Anaya, R. (1999). Un acercamiento a la reutilizacion en ingenieria de software.

Assunção, W., Mendonça, W., and Vergilio, S. (2018). Reúso de software: Do oportunista ao sistemático.

Aydin, S., Steffens, A., and Lichter, H. (2021). Automated construction of continuous delivery pipelines from architecture models. In *28th Asia-Pacific Software Engineering Conference, APSEC 2021, Taipei, Taiwan, December 6-9, 2021*, pages 306–316. IEEE.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Publishing Company.

Booch, G. (1990). *Object Oriented Design with Applications*. Benjamin-Cummings Publishing Co., Inc., USA.

Bosch, J. and Svahnberg, M. (2002). On the notion of variability in software product lines a maintainability analysis of dependability evaluation of an avionics system using aadl to pnml transformation view project managing architectural technical debt view project jilles van gurp formation gmbh.

Chen, T., Zhang, Y., Chen, S., Wang, T., and Wu, Y. (2021). Let's supercharge the workflows: An empirical study of github actions. In *21st IEEE International Conference on Software Quality, Reliability and Security, QRS 2021 - Companion, Hainan, China, December 6-10, 2021*, pages 1–10. IEEE.

Cusumano, M. A. and Selby, R. W. (1998). *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*. Simon & Schuster Trade.

Debroy, V., Miller, S., and Brimble, L. (2018). Building lean continuous integration and delivery pipelines by applying devops principles: A case study at varidesk. *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*.

DORA, D. R. . A. (2021). *Accelerate State of DevOps 2021*. DevOps Research & Assessment DORA.

Duvall, P., Matyas, S., and Glover, A. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional, first edition.

Gallaba, K. (2021). *Improving the Robustness and Efficiency of Continuous Integration and Deployment*. PhD thesis, McGill University, 3480 Rue University, Montréal, QC, Canada.

Gallaba, K. and McIntosh, S. (2020). Use and Misuse of Continuous Integration Features: An Empirical Study of Projects that (mis)use Travis CI. *IEEE Transactions on Software Engineering (TSE)*, 46(1):33–50.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns, Elements of Reusable Object-Oriented Software*. Pearson Education Corporate Sales Division.

Humble, J. and Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 1st edition.

Ivanov, V. and Smolander, K. (2018). Implementation of a devops pipeline for serverless applications. *Lecture Notes in Computer Science*, 11271 LNCS:48–64.

Jin, X. (2022). *Cost-saving in Continuous Integration: Development, Improvement, and Evaluation of Build Selection Approaches*. PhD thesis, Virginia Polytechnic Institute and State University.

Jones, C. (2018). Using code generation to enforce uniformity in software delivery pipelines. In Bruel, J., Mazzara, M., and Meyer, B., editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment - First International Workshop, DEVOPS 2018, Chateau de Villebrumier, France, March 5-6, 2018, Revised Selected Papers*, volume 11350 of *Lecture Notes in Computer Science*, pages 155–168. Springer.

Jumani, A. K., Shaikh, A. A., Khan, M. O., and Siddique, W. A. (2020). Fast delivery, continuously build, testing and deployment with devops pipeline techniques on cloud. *Indian Journal of Science and Technology*, 13:552–575.

Karlsson, E.-A., Andersson, L.-G., and Leion, P. (2000). Daily build and feature development in large distributed projects. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE '00, page 649–658, New York, NY, USA. Association for Computing Machinery.

Kinsman, T., Wessel, M. S., Gerosa, M. A., and Treude, C. (2021). How do software developers use github actions to automate their workflows? In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17-19, 2021*, pages 420–431. IEEE.

Labouardy, M. (2021). *Pipeline as Code*. Manning Publications Co.

Lachhman, R. (2021). *Pipeline Patterns for CI/CD*.

Mäkitalo, N., Taivalsaari, A., Kiviluoto, A., Mikkonen, T., and Capilla, R. (2020). On opportunistic software reuse. *Computing*, 102:2385–2408.

Paule, C., Düllmann, T. F., and van Hoorn, A. (2019). Vulnerabilities in continuous delivery pipelines? A case study. In *IEEE International Conference on Software*

Architecture Companion, ICSA Companion 2019, Hamburg, Germany, March 25-26, 2019, pages 102–108. IEEE.

Poppendieck, M. and Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley Longman Publishing Co., Inc., USA.

Priyadarsini, K., Fantin, E., Raj, I., Begum, A. Y., and Shanmugasundaram, V. (2020). Comparing devops procedures from the context of a systems engineer. *Materials Today: Proceedings*.

Renault, O. (2014). Reuse / variability management and system engineering.

TechTarget (2021). E-guide ci/cd pipelines explained: Everything you need to know.

Vassallo, C., Proksch, S., Jancso, A., Gall, H. C., and Penta, M. D. (2020). Configuration smells in continuous delivery pipelines: a linter and a six-month study on gitlab. In Devanbu, P., Cohen, M. B., and Zimmermann, T., editors, *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, pages 327–337. ACM.

vom Brocke, J., Hevner, A., and Maedche, A. (2020). Introduction to design science research. pages 1–13.

Zampetti, F., Vassallo, C., Panichella, S., Canfora, G., Gall, H. C., and Penta, M. D. (2020). An empirical characterization of bad practices in continuous integration. *Empir. Softw. Eng.*, 25(2):1095–1135.