

19



OFICINA ESPAÑOLA DE
PATENTES Y MARCAS

ESPAÑA



11 Número de publicación: **2 989 585**

51 Int. Cl.:

G06F 8/70 (2008.01)

G06F 21/57 (2013.01)

G06F 21/56 (2013.01)

G06N 5/022 (2013.01)

G06Q 10/0631 (2013.01)

12

TRADUCCIÓN DE PATENTE EUROPEA

T3

86 Fecha de presentación y número de la solicitud internacional: **18.11.2019** **PCT/CA2019/051639**

87 Fecha y número de publicación internacional: **28.05.2020** **WO20102884**

96 Fecha de presentación y número de la solicitud europea: **18.11.2019** **E 19887688 (0)**

97 Fecha y número de publicación de la concesión europea: **25.09.2024** **EP 3884378**

54 Título: **Automatización de identificación de tareas en un ciclo de vida del software**

30 Prioridad:

19.11.2018 US 201862769091 P

45 Fecha de publicación y mención en BOPI de la
traducción de la patente:
27.11.2024

73 Titular/es:

SECURITY COMPASS TECHNOLOGIES LTD.
(100.0%)

1 Yonge Street, Suite 1801
Toronto, ON M5E 1W7, CA

72 Inventor/es:

BHALLA, NISHCHAL;
SETHI, ROHIT KUMAR;
SIVARANJAN, RAMANAN;
FOROUGH, EHSAN y
WHITTINGTON, GEOFFREY CHARLES

74 Agente/Representante:

CURELL SUÑOL, S.L.P.

ES 2 989 585 T3

Aviso: En el plazo de nueve meses a contar desde la fecha de publicación en el Boletín Europeo de Patentes, de la mención de concesión de la patente europea, cualquier persona podrá oponerse ante la Oficina Europea de Patentes a la patente concedida. La oposición deberá formularse por escrito y estar motivada; sólo se considerará como formulada una vez que se haya realizado el pago de la tasa de oposición (art. 99.1 del Convenio sobre Concesión de Patentes Europeas).

DESCRIPCIÓN

Automatización de identificación de tareas en un ciclo de vida del *software*

5 Referencia cruzada a solicitudes relacionadas

Esta solicitud reivindica prioridad respecto a la solicitud de patente provisional de Estados Unidos US62/769.091 presentada el 19 de noviembre de 2019.

10 Campo de la invención

La presente invención se refiere a un sistema y a un método para la automatización de la identificación y el control de tareas en un entorno de ciclo de vida del *software*. En particular, el presente sistema y método es para automatizar la identificación de contexto de *software* durante el desarrollo y el funcionamiento, y la priorización y orquestación de tareas durante un ciclo de vida del *software*.

Antecedentes

El proceso del ciclo de vida de desarrollo de *software* (SDLC) es un proceso para planificar, crear, probar e implementar un sistema de información o un activo de *software* a lo largo de todo el ciclo de vida del activo de *software*. El SDLC comprende la investigación, el análisis, el diseño, la implementación y el mantenimiento desde el principio hasta el fin de un activo de *software*. El desarrollo de nuevo *software* a menudo implica la incorporación de código, bibliotecas y arquitectura de red existentes, cada uno de los cuales viene con su propio conjunto de vulnerabilidades, algunas de las cuales se conocen y otras se desconocen o aún no están identificadas. Identificar las características y los requisitos de seguridad necesarios para cualquier *software* dado y garantizar que los desarrolladores aborden adecuadamente cada requisito durante el ciclo de vida del *software* es una tarea compleja a menudo mediada por una multitud de herramientas independientes de gestión del ciclo de vida de las aplicaciones (ALM) y de gestión de proyectos.

Con el panorama de seguridad electrónica en constante cambio, el *software* requiere una actualización constante para reducir los riesgos de seguridad y prevenir infracciones. Con la integración de *software* en una pluralidad de recursos internos y externos, la identificación de requisitos durante la duración del ciclo de vida del *software* es una tarea onerosa, y mantenerse al día de los requisitos, la vulnerabilidad, las actualizaciones y las infracciones también puede ser oneroso durante el ciclo de vida de desarrollo y mantenimiento del *software*. Los analistas, desarrolladores y evaluadores de *software* pueden verse abrumados por la cantidad de información disponible y la variedad de herramientas que pueden emplear, tener que considerar largas listas de requisitos de tarea, directrices y normas, y tener que proporcionar evidencia tangible y auditable de que los productos de *software* cumplen con los requisitos. Las bibliotecas, recursos y secuencias de comandos vinculados externamente incorporados a un activo de *software* también pueden ser dinámicos y distribuirse automáticamente, de modo que las actualizaciones y las infracciones pueden no llamar inmediatamente la atención de los desarrolladores. El error humano por omisión inadvertida de un requisito o verificación insuficiente también puede introducir vulnerabilidades en el activo de *software*.

A pesar de que las vulnerabilidades de *software* conocidas son de conocimiento público tanto para los desarrolladores como para los piratas informáticos, los desarrolladores de *software* a menudo carecen de herramientas y orientación relevantes, oportunas y específicas del contexto que los ayuden a construir y mantener *software* seguro. Muchas herramientas se centran únicamente en detectar una vulnerabilidad específica en el código fuente de un *software* y están diseñadas para analizar el código fuente y/o versiones compiladas del código para ayudar a encontrar fallas de seguridad. Sin embargo, corregir una vulnerabilidad después de la codificación es costoso y a menudo difícil, y la identificación y priorización de tareas procedentes de recursos de fuentes externas puede ser un desafío para los desarrolladores. Los grandes repositorios de información regulatoria y de seguridad pueden ser difíciles de consultar, no están personalizados para un entorno de aplicación específico y, a menudo, no están sujetos a controles de calidad de nivel comercial. Presentar múltiples repositorios de requisitos de gran tamaño y en constante evolución también requiere que los desarrolladores no solo identifiquen todos y cada uno de los requisitos relativos a una aplicación en particular, sino que también se mantengan actualizados con nueva información y requisitos.

Se han realizado algunos intentos para determinar los riesgos de defectos en las soluciones de *software* durante el desarrollo. La patente de Estados Unidos 9.558.464 concedida a Bassin *et al.* enseña un sistema y un método para recibir uno o más factores de riesgo, recibir uno o más contextos, identificar una o más relaciones de contexto y asociar el uno o más contextos con uno o más factores de riesgo, donde el uno o más factores de riesgo mapean un contexto asociado a una consecuencia de riesgo relacionada con un defecto de *software* para determinar un modelo de riesgo. Entonces se ejecutan pruebas basadas en el riesgo según el modelo de riesgo para determinar una evaluación del riesgo relacionado con el defecto para un proyecto de desarrollo de *software*.

Otras herramientas han intentado resolver problemas de seguridad de *software* proporcionando herramientas tales

como análisis estático de *software* y aplicaciones de pruebas en tiempo de ejecución. La solicitud de patente de Estados Unidos 2012/0254829 concedida a Bhalla *et al.* describe un método para proporcionar una orientación de seguridad al escribir activos de *software* que incluye la activación de una aplicación de orientación vinculada a un ordenador y una base de datos de características de seguridad, pudiendo hacerse funcionar la aplicación de orientación para presentar a un usuario contenido de seguridad sugerente al escribir activos de *software*.

La solicitud de patente de Estados Unidos US2009/083695A1 concedida a Mir *et al.* describe un sistema y un método para el análisis y modelado de amenazas a empresas mediante la definición de una aplicación de *software* y sus atributos y reglas relacionadas con la aplicación y la generación de una lista de tareas a partir de una lista de tareas comunes para la aplicación. La solicitud de patente internacional PCT WO2017/134665 concedida a Mizrahi *et al.* describe un sistema y un método para analizar una colección de código fuente mediante el análisis sintáctico de los datos en una arquitectura de árboles de sintaxis abstracta utilizando grafos de código, ontología de tipos y características semánticas para organizar, indexar y construir un motor de búsqueda de código fuente empleando especificaciones de lenguaje natural. Cadariu Mircea *et al.*: "Tracking Known Security Vulnerabilities in Proprietary Software Systems", 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE, 2 de marzo de 2015, páginas 516-519 describe un proceso basado en herramientas para realizar un seguimiento de las vulnerabilidades conocidas en sistemas de *software* a lo largo de todo su ciclo de vida.

Sigue existiendo la necesidad de una extracción automatizada de los requisitos de tarea y seguridad para un activo de *software* basándose en el contexto de *software* para automatizar la identificación de tareas en un ciclo de vida del *software*.

Esta información de antecedentes se proporciona con el fin de divulgar información que el solicitante cree que puede ser de posible relevancia para la presente invención. No se pretende admitir necesariamente, ni debe interpretarse, que alguna de las informaciones precedentes constituye técnica anterior frente a la presente invención.

Sumario de la invención

Un objetivo de la presente invención es proporcionar un sistema y un método para la extracción automatizada de requisitos de tarea y seguridad para un activo de *software* basándose en el contexto de proyecto de *software* para guiar el desarrollo de *software* y para identificar los requisitos de seguridad del activo de *software* que deben abordarse a lo largo de todo el ciclo de vida del *software*.

La presente invención se define mediante las reivindicaciones independientes. Las formas de realización de la invención se describen en las reivindicaciones dependientes.

Breve descripción de las figuras

Para una mejor comprensión de la presente invención, así como de otros aspectos y características adicionales de la misma, se hace referencia a la siguiente descripción que se utilizará conjuntamente con los dibujos adjuntos, donde:

la figura 1 ilustra un sistema para generar una lista de tareas priorizadas mediante la extracción de contexto de *software* de un repositorio de contexto;

la figura 2 ilustra un sistema para el almacenamiento y la extracción de contexto;

la figura 3 ilustra un método de flujo de datos desde repositorios de contexto en el presente sistema;

la figura 4 ilustra un método de identificación y verificación automatizadas de tareas en un ciclo de vida del *software*;

la figura 5 ilustra un método para extraer datos de contexto para un proyecto y seleccionar requisitos basándose en el contexto extraído;

la figura 6 ilustra un método para orquestar el flujo de trabajo para sistemas humanos o automatizados; y

la figura 7 ilustra un flujo de trabajo para incorporar uno o más proyectos.

Descripción detallada de la invención

A menos que se definan de otro modo, todos los términos técnicos y científicos utilizados en la presente memoria presentan el mismo significado que el que entiende habitualmente un experto en la materia a la que pertenece esta invención.

Tal como se utilizan en la memoria y las reivindicaciones, las formas en singular “un”, “una” y “el/la” incluyen referencias en plural a menos que el contexto indique claramente otra cosa.

- 5 El término “que comprende” tal como se utiliza en la presente memoria se entenderá que significa que la lista que sigue no es exhaustiva y puede o no incluir otros elementos adicionales adecuados, por ejemplo, una o más característica(s), componente(s) y/o elemento(s) adicional(es), según corresponda.

10 El término “desarrollador” tal como se utiliza en la presente memoria se refiere a cualquier persona implicada en el desarrollo, mantenimiento, pruebas y/o aspectos regulatorios de un activo de *software* en cualquier momento durante el desarrollo de ciclo de vida del *software*. Ejemplos no limitativos de desarrolladores incluyen desarrolladores de *software*, analistas de requisitos, programadores, evaluadores, jefes de proyectos, auditores, asesores regulatorios y analistas de seguridad.

15 El término “debilidad” tal como se utiliza en la presente memoria se refiere a un riesgo de seguridad general que puede producirse en un *software* y que potencialmente puede permitir que un atacante y/o un código o acción disruptiva interfieran con el *software* o lo infrinjan. Una debilidad puede ser consecuencia, por ejemplo, de una susceptibilidad o falla del sistema, el acceso de un atacante a la falla, la capacidad de un atacante para aprovecharse de la falla, la susceptibilidad a la adición no autorizada de código, la susceptibilidad a la interrupción del *software* o una combinación de los mismos.

20 El término “vulnerabilidad” tal como se utiliza en la presente memoria se refiere a la aparición particular de una debilidad en un *software*, ya sea en el código o en la aplicación del código, y que presenta un riesgo de seguridad para el *software* en particular.

25 El término “activo de *software*” se refiere a cualquier sistema intensivo de *software* que permite o realiza cálculos. Los ejemplos de activos de *software* incluyen, pero no se limitan a, un lenguaje de programación, un código fuente, una aplicación de *software*, una base de datos, un sistema operativo, un ordenador de sobremesa, un servidor o un dispositivo de informático o de comunicación de hardware.

30 El término “tarea” se refiere a cualquier control o requisito con fines de cumplimiento, obtención de una funcionalidad o abordar un riesgo. El requisito puede ser un requisito no funcional (NFR) para un sistema o *software* que está diseñándose o desarrollándose, y/o un requisito que especifica los criterios que pueden utilizarse para juzgar el funcionamiento y/o el riesgo de un sistema. Los ejemplos no limitativos de requisitos no funcionales incluyen accesibilidad, adaptabilidad, auditabilidad, disponibilidad, capacidad, certificación, compatibilidad, gestión de configuración, integridad de datos, retención de datos, dependencia, implementación, entorno de desarrollo, recuperación ante desastres, documentación, durabilidad, eficiencia (consumo de recursos para una carga dada), explotabilidad, extensibilidad, gestión de fallos, tolerancia a fallos (por ejemplo, monitorización, medición y gestión del sistema operativo), cuestiones jurídicas y de concesión de licencias o evitación de infracciones de patentes, interoperabilidad, mantenibilidad (por ejemplo, tiempo medio de reparación - MTTR), gestión, modificabilidad, topología de red, operabilidad, rendimiento y/o tiempo de respuesta (ingeniería de rendimiento), compatibilidad de plataformas, privacidad (cumplimiento de las leyes de privacidad), portabilidad, calidad (por ejemplo, fallos descubiertos, fallos detectados, eficacia de eliminación de fallos), legibilidad, fiabilidad (por ejemplo, tiempo medio entre/hasta fallos - MTBF/MTTF), resiliencia, limitaciones de recursos (velocidad del procesador, memoria, espacio en el disco, ancho de banda de red, etc.), tiempo de respuesta, reusabilidad, robustez, escalabilidad (horizontal, vertical), seguridad (cibernética y física), estabilidad, capacidad de soporte, capacidad de prueba, rendimiento, transparencia y usabilidad (factores humanos) por comunidad de usuarios objetivo.

50 El término “datos de contexto” se refiere a cualquier dato perteneciente a un activo de *software* que sea o contribuya a una propiedad, característica o condición técnica u operativa del activo de *software*. Los datos de contexto pueden inferirse o extraerse de un repositorio de contexto del activo de *software*. Los datos de contexto pueden ser técnicos o no técnicos e incluyen, pero no se limitan a, código fuente, arquitectura del sistema, ubicaciones de archivo y tipos de archivos, descripciones en lenguaje natural de las características y componentes deseados del activo de *software*, y cualquier dato extraído de ellos que pertenezca al activo de *software*. Los datos de contexto pueden obtenerse de cualquier repositorio que almacene datos pertenecientes al activo de *software* incluyendo, pero sin limitarse a, sistemas de registro, código fuente, especificaciones de *software*, sistemas y bases de datos, sistemas de gestión de dependencias, *software* de gestión de proyectos, *software* de gestión de negocio, sistemas de gestión de código fuente (SCM), sistemas de gestión de configuración, sistemas de gestión del ciclo de vida de las aplicaciones, planes de prueba, bibliotecas de dependencia y cualquier derivado de los datos contenidos en estos sistemas, y pueden incluir descripciones textuales del activo de *software* o su funcionalidad en lenguaje natural o sencillo.

65 En la presente memoria, se describe un sistema y un método para la identificación automatizada de tareas y la extracción de requisitos de tarea para un activo de *software* basándose en el contexto de proyecto de *software*. El método y el sistema descritos actualmente también proporcionan orientación para el desarrollo de *software* seguro basándose en el contexto de *software* y dirige la atención de los desarrolladores a los requisitos de tarea y de

seguridad del *software* que deben abordarse durante todo el ciclo de vida de desarrollo de *software*.

El presente método y sistema es aplicable al desarrollo de *software* durante todo el ciclo de vida de desarrollo de *software* para identificar los requisitos de tarea utilizando datos de contexto extraídos de por lo menos un repositorio de contexto para el activo de *software* y para priorizar los requisitos de tarea durante todas las fases del ciclo de vida del *software*. Para automatizar tareas en un activo de *software*, el sistema genera una guía práctica para orquestar el desarrollo de *software* a través de una lista de tareas priorizadas y guía el desarrollo de una aplicación individual y una cartera de *software* basándose en el contexto de cada activo informático en la cartera. Los procedimientos automáticos que pueden abordar eficazmente determinadas tareas se dirigen a implementar la orientación y se lleva a cabo una verificación del control de riesgos para confirmar que se han abordado los requisitos de tarea identificados. El sistema puede mejorar adicionalmente basándose en señales detectadas en cada etapa del procedimiento mediante el aprendizaje automático. Al integrar las bases de datos de contexto del activo de *software* con la extracción automatizada del contexto de *software* y una base de datos de conocimiento, el presente sistema y método puede proporcionar una lista priorizada de la lista de requisitos de tarea para guiar el trabajo de los desarrolladores para optimizar el desarrollo y el mantenimiento de *software* y mitigar el riesgo en un sistema en todas las etapas del ciclo de vida del *software*.

A modo de ejemplo, una organización de negocio presenta una multitud de activos de *software* en diversos estados de ideación, desarrollo e implementación. Cada uno de estos activos de *software* presenta su propio conjunto de características, utiliza su propio conjunto de bibliotecas, presenta un sistema de archivos único con ubicaciones de archivos, presenta una multitud de archivos cada uno con su propio lenguaje y código, y presenta su propio conjunto de riesgos dependiendo de todos estos factores. Todas las características y condiciones que contribuyen a la estructura y el funcionamiento de cada activo de *software* se denominan en la presente memoria el contexto de *software*, y el contexto de *software* contribuye a y determina los riesgos que plantea el activo de *software* después de su implementación. Durante la fase de ideación de un proyecto para crear un activo de *software*, el contexto del activo de *software* puede limitarse a las características deseadas por una unidad de negocios y el contexto puede escribirse completamente en lenguaje natural. El contexto de *software* en la fase de predesarrollo a menudo se almacena como una o más historias o epopeyas (historias largas) en una plataforma de organización de negocio, donde las historias habitualmente son descripciones breves y sencillas de una característica o un conjunto de características contadas desde la perspectiva de la persona que desea la nueva capacidad, habitualmente un usuario o cliente del sistema. Las historias normalmente siguen una plantilla sencilla, tal como "Como <tipo de usuario>, quiero <algún objetivo> de modo que <algún motivo>", donde el usuario describe habitualmente el motivo en lenguaje natural para definir los términos de la característica, función o resultado deseados. El escritor de las historias a menudo es un usuario del sistema y no un desarrollador, por lo que las solicitudes de características suelen escribirse en lenguaje natural. Las organizaciones de negocio desarrollan listas de proyectos futuros a partir de estas historias y seleccionan proyectos para desarrollar basándose en sus prioridades de negocio. El contexto de *software* puede obtenerse a partir de estas historias mediante el procesamiento del lenguaje natural para extraer información como las características deseadas, la jurisdicción de la utilización de *software*, el caso de utilización del *software*, la población de utilización del *software* y otro contexto relacionado con el funcionamiento del *software* y el entorno de seguridad. Las listas de proyectos pueden almacenarse, por ejemplo, en sistemas internos de registro o bases de datos o *software* de gestión de proyectos que comprenden sistemas básicos de planificación, ejecución, contabilidad de proyectos y análisis pertenecientes a los activos de *software* de negocio. En el desarrollo inicial de un proyecto de *software*, se recurre a los repositorios de contexto para proporcionar precedentes y bibliotecas, todo lo cual contribuye al contexto de *software*. Además, pueden introducirse riesgos en la utilización de estos repositorios de contexto, o bien mediante la incorporación de una vulnerabilidad en el momento del desarrollo o bien mediante la utilización o la integración de una biblioteca dinámica o parte de ella, en la que se introduce una vulnerabilidad posteriormente y el conocimiento de la vulnerabilidad introducida solo puede llamar la atención de los desarrolladores si el flujo de trabajo del activo de *software* identifica y considera el riesgo de esa biblioteca vulnerable a lo largo de todo el ciclo de vida del *software*. Mediante la utilización de métodos automatizados para extraer información de contexto sobre el activo de *software* y a través de la actualización regular de la priorización de riesgos, incluyendo la vigilancia para asesorar a los desarrolladores sobre posibles vulnerabilidades introducidas por los repositorios de contexto vinculados, puede proporcionarse a los desarrolladores un plan priorizado para el desarrollo de *software* para identificar y mitigar los riesgos durante el ciclo de vida del *software*.

El contexto de *software* encapsula dominios técnicos y puede encapsular dominios de negocio o de procedimiento, y puede comprender reglas de inclusión relacionadas con reglas utilizadas en otras partes del sistema. Las reglas pueden estar compuestas de operadores booleanos y propiedades del proyecto, por ejemplo, la aplicación utiliza Java Y la aplicación está basada en la web. Basándose en el contexto del activo de *software*, se identifican tareas, riesgos y requisitos de seguridad a partir de una base de datos de conocimiento y se genera una lista de tareas priorizadas y se presenta a los desarrolladores para que la aborden durante el desarrollo de *software*, durante la implementación o después de la implementación. La consideración dinámica del contexto de *software* puede realizarse a lo largo de todo el ciclo de vida del *software* para proporcionar a los desarrolladores requisitos de seguridad a medida que surgen, incluyendo orientación sobre cómo abordar cada requisito. De este modo, la seguridad y la prevención de riesgos, así como la orquestación de tareas, pueden integrarse en el activo de *software* desde las fases iniciales del ciclo de vida del *software* a través de la identificación automatizada del

contexto de *software* y mantenerse durante toda su vida útil. Muchos proyectos pueden incorporarse automáticamente utilizando repositorios de contexto para extraer contexto sobre un nuevo proyecto, incluyendo cada incorporación datos de contexto extraídos de repositorios de contexto de confianza, y muchos sistemas y repositorios de contexto existentes pueden aprovecharse para proporcionar amplitud de contexto y actualización dinámica de tareas durante todo el ciclo de vida del activo de *software*. En un ejemplo que no es según la invención, la extracción del contexto de *software* desde los repositorios de contexto de lenguaje natural también puede ayudar a proporcionar información adicional a los responsables de la toma de decisiones de negocio con respecto al esfuerzo y el coste de añadir una característica o desarrollar un nuevo activo de *software* basándose en los requisitos y tareas necesarias para lograr un nivel aceptable de funcionalidad y/o riesgo, lo que puede ayudar adicionalmente con la toma de decisiones de negocio y la priorización de nuevos proyectos de *software*.

La figura 1 ilustra un sistema 100 para generar una lista de tareas priorizadas para un activo de *software* mediante la extracción de contexto de *software* de un repositorio de contexto. El sistema utiliza el contexto de *software* en cada repositorio de contexto 102 para seleccionar tareas o requisitos a partir de una base de datos de conocimiento 108 basándose en el contexto de *software* extraído. El sistema proporciona la identificación automatizada de tareas en un activo de *software* basándose en el contexto de *software* extraído a lo largo de todo el ciclo de vida del *software*, de modo que los requisitos de tarea puedan identificarse y actualizarse a lo largo de toda la vida útil del activo de *software*.

Uno o más repositorios de contexto 102 son bases de datos que contienen información de contexto sobre el activo de *software* y comprenden datos de contexto que pertenecen a la construcción y/o funcionamiento del *software*, donde los datos de contexto comprenden las características del entorno de *software* que pertenecen a la descripción, codificación y ejecución del activo de *software*. En una organización, los especialistas de negocio a menudo utilizan bases de datos o sistemas de lenguaje natural para registrar y solicitar características o *software* que harán avanzar a la organización, y los repositorios de contexto son cualquier sistema donde puede encontrarse el contexto específico para un activo de *software* nuevo o existente en una organización.

El contexto de *software* describe las características del activo de *software* incluyendo, pero sin limitarse a, el lenguaje, la función, los requisitos regulatorios y las conexiones del activo de *software*. En un ejemplo, una aplicación o sistema puede describirse como poseedor de un conjunto de atributos que describen su comportamiento y estado, lo que se define más ampliamente como su "contexto". Considérese el conjunto C que contiene el contexto de *software* para un activo de *software*, donde C se compone de n formas posibles de describir un sistema o aplicación, y cada elemento del conjunto es un contexto técnico, de negocio u otro contexto relevante para el sistema. En un ejemplo, el conjunto de datos de contexto C puede definirse como:

$$C = \{c_1, \dots, c_i, \dots, c_n; 1 < i < n\},$$

donde cada elemento de contexto c_i contiene comportamiento o estado de contexto sobre el activo o la aplicación de *software*,

y los valores c_i pueden ser:

c_1 : El funcionamiento y el tiempo de ejecución de la aplicación están sujetos a las leyes de los Estados Unidos.

c_2 : La aplicación está desarrollada utilizando el lenguaje de programación Java.

c_3 : La aplicación utiliza una base de datos

c_4 : La aplicación debe estar libre de vulnerabilidades de alta prioridad antes de poder implementarse en producción.

Las categorías de contexto de *software* pueden comprender, por ejemplo, la tecnología utilizada, los datos que se manejan, las fuentes de cumplimiento legales y otras, los atributos del entorno tales como factores geográficos y jurisdiccionales, los requisitos funcionales que afectan a los aspectos de seguridad, las dependencias, el lenguaje de programación, la plataforma, el área de tecnología del activo de *software*, los datos recopilados por el activo de *software*, los datos compartidos por el activo de *software*, el dominio de negocio, los requisitos regulatorios, las normas de seguridad, la arquitectura de red, los datos procesados o emitidos, el contexto legal (como dónde o cómo se utiliza la aplicación), la plataforma operativa y el contexto personalizado descrito por un usuario final. El contexto de *software* también puede comprender la naturaleza del proyecto, las características de seguridad, los usuarios de la aplicación, las configuraciones de restricción de acceso, los tipos de información que maneja la aplicación, los impulsores y las políticas de negocio, las plataformas, los requisitos de privacidad, las tecnologías, los lenguajes de programación, las interfaces de aplicación, el tipo de aplicación, el tipo de servidor(es) web utilizado(s), las tecnologías y bibliotecas de terceros integradas, la industria, el entorno, los requisitos tecnológicos y otras tecnologías integradas. Mediante la extracción y el análisis de la información de contexto sobre el activo de *software*, puede recuperarse automáticamente un conjunto personalizado de tareas y requisitos de la base de

datos de conocimiento basándose en su relevancia para el proyecto de *software* específico.

El motor de extracción de contexto 104 procesa cada repositorio de contexto 102 para transformarlo en información utilizable por máquina y para extraer datos de contexto para el activo de *software*. El motor de extracción de contexto 104 puede comprender escáneres de código, herramientas de procesamiento de lenguaje natural, aprendizaje automático, y otras herramientas. El metaanálisis de los datos de contexto extraídos por el motor de extracción de contexto 104 genera información sobre el activo de *software*, incluyendo la estructura del archivo o la fuente del sistema del activo de *software*. El metaanálisis puede proporcionar información específica tal como, por ejemplo:

Nombres de archivo: una lista de todos los archivos utilizados para describir e implementar el estado y el comportamiento de una aplicación o sistema

Tipos de archivos: una lista de todos los tipos de archivos distintos que utiliza la aplicación o el sistema, así como un recuento

Registro de cambios: la lista de cambios realizados en la definición de fuente, incluyendo quién o qué realizó el cambio, la fecha/hora en que se realizó el cambio y una descripción textual relacionada con el cambio

Archivos de manifiesto/configuración: detalles sobre el estado o el comportamiento de una aplicación o un sistema que están codificados en uno o más archivos fuente (por ejemplo, archivos .ini, .xml, .sql, .json) que contienen identificadores o detalles para inicializar la aplicación/el sistema, o instrucciones sobre cómo debe comportarse una aplicación.

Dependencias: La lista de componentes autónomos de los que depende una aplicación o un sistema para funcionar correctamente. Conocer que la dependencia X es parte de una aplicación o sistema implica información sobre su comportamiento y estado, por ejemplo, si se sabe que una biblioteca interactúa con los datos del Reglamento General de Protección de Datos (GDPR) europeo, si se observan las dependencias, entonces el contexto de aplicación incluye las normas de requisitos del GDPR.

El contexto de *software* también puede incluir información relacionada con la liberación o el cambio del activo de *software*, los desarrolladores o equipos de desarrolladores que trabajaron en el proyecto, las respuestas a las consultas de los usuarios, los nombres o identificadores del activo de *software* y/o la identificación de la unidad de negocio responsable de la aplicación. El contexto de *software* también puede incluir aspectos técnicos del activo de *software* tales como, por ejemplo, integraciones del sistema de gestión configuradas para la aplicación, integraciones del escáner configuradas para la aplicación y detalles de la política de riesgos. En un ejemplo, considérese una aplicación PHP, que podría presentar un perfil de metaanálisis tal como se muestra en la tabla 1:

Tabla 1

Atributo	Valor
Archivos	• index.php • composer.json • index.css • Init.sql • Update.sql • App.cfg
Tipos de archivo	• .php (recuento = 1) • .json (recuento = 1) • .sql (recuento = 2) • .json (recuento = 1) • .cfg (recuento = 1)
Mensajes de confirmación en Git	• John Smith, 1 de junio de 2014, "aplicación creada" • John Smith, 3 de junio de 2014, "lógica añadida para procesar tarjetas de crédito" • John Smith, 8 de junio de 2015 "apoyo añadido para la búsqueda en base de datos"
Manifiesto/configuración	Desde App.cfg: • Detalles de nombre de usuario y contraseña para conectarse a la base de datos de apoyo • Nombre de la aplicación • Nombre de usuario y contraseña de la aplicación
Dependencias	• Libxml • Openssl • Pcre • Postgres

Pueden utilizarse diversas herramientas y sistemas de análisis para obtener datos de contexto a partir de los repositorios de contexto. En un ejemplo particular, si una organización desea construir un activo de *software* para lograr un resultado específico, la organización puede generar tickets que comprendan características de lo que el activo de *software* debería hacer. Los tickets escritos en lenguaje natural pueden convertirse en la entrada para la identificación de tareas en el sistema, donde un procesador de lenguaje natural extrae el contexto del lenguaje natural de los tickets.

El motor de extracción de contexto 104 almacena los datos de contexto que extrae de los repositorios de contexto 102 en una base de datos de contexto 106. La automatización de la identificación y priorización de las tareas operativas y de desarrollo de *software* durante un ciclo de vida de desarrollo de *software* puede realizarse escaneando fuentes de contexto para contexto de *software* utilizando una o más herramientas de raspado apropiadas para escanear cada repositorio de contexto 102. El análisis de datos de contexto raspados o escaneados puede realizarse mediante procesamiento de lenguaje natural (NLP).

También pueden utilizarse aprendizaje automático e inteligencia artificial para identificar relaciones entre las características de *software* y el contexto técnico y proporcionar un subconjunto previsto de requisitos de tarea para el activo de *software*, y pueden utilizarse para entrenar el módulo de selección 110 basándose en elementos de tareas que ya se encuentran en la base de datos de conocimiento 108, donde pueden extraerse subconjuntos de tareas relacionadas como un grupo relacionado y añadirlos a la lista de tareas. Por ejemplo, si la extracción de contexto revela que se requiere información de tarjeta de crédito Y la jurisdicción operativa de un activo de *software* es Francia, el conjunto de requisitos relevantes para {función: tarjeta de crédito} y {jurisdicción: Francia} puede extraerse automáticamente y añadirse a la lista de tareas. En otro caso de identificación del contexto de aplicación, el motor de extracción de contexto puede extraer datos de contexto consultando un conjunto de identidades de datos de contexto, por ejemplo:

{Archivos .java, dependencia de base de datos, ID de aplicación interna}

Un procesador de contexto a su vez extraería el siguiente contexto para el activo de *software*:

Detalles del proyecto del activo de *software* = {
respuestas de la encuesta del proyecto: "Perfil web Java", "Utiliza una base de datos"
atributos personalizados: ID de aplicación interna
}

Las herramientas y sistemas de análisis también pueden proporcionar información sobre vulnerabilidades y amenazas acerca de una aplicación, así como sobre las características del producto. Por ejemplo, si Fortify encuentra una vulnerabilidad de "inyección SQL", se supone que el activo de *software* está utilizando una base de datos, ya que la debilidad "inyección SQL" solo puede producirse si hay una base de datos en utilización. El contexto rastreado por estas herramientas y sistemas puede describirse como un conjunto de t elementos de contexto, donde:

$$C = \{c_1, \dots, c_i, \dots, c_t\}$$

donde c_i es un hecho de análisis sobre un activo de *software*.

Por ejemplo, un escáner de código tal como Fortify™ realiza un seguimiento de hechos de análisis c_i sobre una aplicación donde c_i es una debilidad detectada en la aplicación, tal como mediante inyección SQL o por Transporte Inseguro: Tiempo de Expiración de HSTS Insuficiente. El conjunto C también puede incluir m datos de contexto recopilados a partir de herramientas de la fase de implementación, tales como los sistemas de detección de intrusiones (IDS), en los que:

$$C \{c_1, \dots, c_d, \dots, c_m\}$$

donde c_d es una amenaza de aplicación detectada, tal como, por ejemplo, "intento de respuesta TCP sobre DNS" o "intento de ataque de escalada de privilegios de la aplicación".

La información de contexto de *software* que puede obtenerse automáticamente de fuentes electrónicas puede utilizarse para generar la lista de tareas. Cuando los datos de contexto extraídos son inciertos, el sistema puede interrogar a un desarrollador para ampliar cualquier información necesaria para completar la lista de tareas. La base de datos de conocimiento 108 es un repositorio de conocimiento de directrices de tareas accesibles con una herramienta de recuperación, y la integración con la base de datos de conocimiento se produce a lo largo de todo el ciclo de vida del *software*, tal como durante la creación del proyecto, durante la fase de requisitos, la fase de diseño, la fase de desarrollo, la fase de prueba, la fase de implementación, la fase de mantenimiento y actualización, y la fase de reemplazo y obsolescencia o depreciación. La base de datos de conocimiento 108 contiene información de requisitos de tarea para una amplia variedad de aplicaciones de *software*, contextos e implementaciones, y se actualiza regularmente para adaptarse a los requisitos en tiempo real y los desafíos de

seguridad. La base de datos de conocimiento 108 comprende una pluralidad de tareas, presentando cada tarea una identificación de un requisito de codificación, un requisito de seguridad o un requisito regulatorio, y una estrategia recomendada para completar el requisito de codificación, un mecanismo para abordar el requisito regulatorio o la solución del requisito de seguridad y controles de mitigación. Las tareas en la base de datos de conocimiento 108 pueden referirse, por ejemplo, a requisitos de seguridad generales relevantes, directrices sobre el diseño de arquitectura segura, normas de desarrollo generales para diferentes plataformas y tecnologías de desarrollo, código de muestra sometido a prueba para implementar las normas, listas de casos de pruebas de seguridad y/o listas de comprobación para permitir que los desarrolladores no expertos sometan a prueba sistemáticamente los requisitos de seguridad, módulos de enseñanza, seguimiento de la gestión del ciclo de vida de las aplicaciones y herramientas de integración de seguimiento de errores. En las directrices de tareas de la base de datos de conocimiento 108 también pueden incluirse módulos de enseñanza y enlaces a requisitos de cumplimiento regulatorio, incluyendo la norma de seguridad de datos de la industria de tarjetas de pago (PCI DSS), la ley de portabilidad y responsabilidad de seguros médicos (HIPPA), la ley de tecnología de la información sanitaria para la salud económica y clínica (HITECH), la ley Gramm-Leach-Bliley (GLBA), la ley de protección de infraestructuras críticas de la corporación norteamericana de fiabilidad eléctrica (North American Electric Reliability Corporation Critical Infrastructure) (NERC CIP), y las leyes internacionales de privacidad, para ayudar a los desarrolladores al reunir todos los requisitos en una sola ubicación. Otras tareas pueden referirse a consejos específicos de tecnología relacionados con normas particulares, por ejemplo, consejos particulares sobre cómo utilizar la biblioteca Anti-XSS de Microsoft para mitigar el Cross Site Scripting (secuencia de comandos en sitios cruzados) para ASP.NET. Además, puede proporcionarse orientación a diferentes partes interesadas en el ciclo de vida de desarrollo de *software* y puede accederse a ella mediante una vista convencional, personalizada o filtrada. Por ejemplo, a los analistas de requisitos se les puede mostrar o pueden ordenar los requisitos de seguridad para el proyecto, los arquitectos de sistemas pueden ver consejos u orientación sobre el diseño de la aplicación, y a los evaluadores se les puede mostrar cómo someter a prueba o validar la finalización de las tareas. También pueden proporcionarse tareas personalizadas y relevantes incluyendo directrices y ejemplos de código para una gran variedad de lenguajes de programación, marcos de trabajo, plataformas y otras tecnologías comunes. Pueden importarse o mostrarse ejemplos de código positivos y ejemplos de código completos para abordar una tarea particular. Los desarrolladores pueden descargar los ejemplos de código y someterlos a prueba o ejecutarlos por sí mismos, por ejemplo, en un entorno de desarrollo integrado (IDE) como Eclipse™, donde los desarrolladores pueden ver el código completo e incluso ejecutar el programa completo para ver la norma en acción.

Cada tarea en la base de datos de conocimiento 108 presenta etiquetas que delinean su relevancia para un contexto de *software* particular, de modo que las tareas requeridas para un activo de *software* particular pueden generarse basándose en los datos de contexto del activo de *software* en la base de datos de contexto 106. El acceso a la base de datos de conocimiento 108 puede producirse a lo largo de todo el proceso de desarrollo y mantenimiento de *software* para proporcionar una lista de tareas priorizadas 114 actualizada y orientación para cada tarea que sea relevante para el activo de *software*. A medida que la base de datos de conocimiento se actualiza dinámicamente, la lista de tareas priorizadas 114 para el activo de *software* o el grupo de activos de *software* también puede actualizarse dinámicamente, de modo que los riesgos se evalúen de forma rutinaria y se incorporen a las normas de orientación del sistema. Sin limitarse a la teoría, uno de los puntos fuertes de la base de datos de conocimiento descrita actualmente es que los cambios en las regulaciones, los requisitos de seguridad, la aparición de *malware* y las actualizaciones de seguridad pueden identificarse y actualizarse rápidamente en el activo de *software* actualizando la base de datos de conocimiento y enviándola a la lista de tareas priorizadas para una gestión adecuada del activo de *software*. De esta manera, los requisitos actualizados pueden priorizarse en la lista de tareas específicas de la aplicación para un activo de *software*. El contexto de *software* que identifica una dependencia que se considera susceptible al *malware* puede escalarse inmediatamente en la lista de tareas prioritarias y mitigarse eficientemente por los desarrolladores. De esta manera, los desarrolladores pueden recibir alertas rápidas sobre actualizaciones relevantes para cada activo de *software* basándose en el contexto de *software* previamente definido y pueden recibir orientación sobre soluciones para mitigar el riesgo en el activo de *software*. También puede accederse directamente a la base de datos de conocimiento 108 mediante una interfaz de usuario con una utilidad de gestión de base de datos. Un servidor web también puede permitir que un usuario autorizado se conecte a una aplicación web por medio de Internet y una interfaz de usuario. Puede utilizarse una interfaz de usuario para codificar el activo de *software* por parte del desarrollador, y puede ser un único dispositivo informático, una pluralidad de ordenadores o dispositivos electrónicos que están utilizándose por una pluralidad de desarrolladores, o una red distribuida de ordenadores que comprenden uno o más procesadores y memorias conectadas operativamente al procesador, y pueden además estar conectados a Internet. La interfaz de usuario también puede presentar acceso a la base de datos de conocimiento 108 y puede ser capaz de acceder a uno o a una pluralidad de proyectos diferentes al mismo tiempo, así como a las listas de tareas priorizadas para uno o más activos de *software* en la organización.

Basándose en el contexto de *software*, se emplea el módulo de selección 110 para seleccionar una pluralidad de requisitos de tarea para el activo de *software* a partir de la pluralidad de requisitos de tarea en la base de datos de conocimiento 108, siendo los requisitos de tarea seleccionados aplicables a los datos de contexto almacenados en la base de datos de contexto 106 para el activo de *software* en desarrollo o mantenimiento. El módulo de selección 110 también puede utilizar métodos informáticos que incluyen aprendizaje automático o inteligencia

artificial para combinar y/o hacer corresponder la información en la base de datos de contexto 106 con las tareas en la base de datos de conocimiento 10 para generar un conjunto de tareas pertinentes al *software* que está considerándose. Los usuarios también pueden presentar la capacidad de procesar la lista de tareas y proporcionar un conjunto de enseñanza al módulo de selección 110 para potenciar el aprendizaje automático y la selección de tareas para proyectos futuros. Por ejemplo, si se requiere la tarea T cada vez que se extrae un determinado contexto en la base de datos, algunos usuarios observarán que la tarea no es aplicable. El módulo de selección 110 puede utilizar entonces los datos de retroalimentación proporcionados por el desarrollador para ayudar en la selección de tareas para proyectos futuros con perfiles de datos de contexto similares. El sistema está preferentemente en funcionamiento durante todo el ciclo de vida del *software*, de modo que las actualizaciones a cualquiera de los repositorios de contexto, la base de datos de conocimiento o el código o arquitectura del *software* se integren y aporten información actualizada y oportuna sobre tareas y requisitos desde la base de datos de conocimiento a la lista de tareas priorizadas.

El motor de priorización 112 prioriza el conjunto de requisitos de tarea seleccionados identificados por el módulo de selección 110, donde la priorización se basa en la orquestación de tareas o el riesgo de cada uno de los requisitos de tarea seleccionados para generar la lista de tareas priorizadas 114. La lista de tareas priorizadas 114 puede almacenarse en un ordenador, servidor o en otro dispositivo o sistema electrónico, y un desarrollador accede a ella y se le presenta en una interfaz de usuario. La lista de tareas priorizadas 114 también puede proporcionar orientación sobre la orquestación de tareas basándose en el cronograma u orden conocido o aprendido más eficiente (en tiempo o dinero) para manejar el conjunto de tareas. La orquestación puede dar como resultado un efecto en cascada en la gestión de tareas, de modo que abordar determinadas tareas tenga un mayor efecto posterior si se identifican en una etapa temprana del proceso de identificación de tareas. Al revisar regularmente la lista de tareas priorizadas 114, los equipos de desarrollo y gestión también pueden obtener visibilidad en tiempo real del estado de verificación y finalización de un proyecto durante la producción. El cronograma de implementación puede evaluarse basándose en las tareas restantes acumuladas y los objetivos de la organización. Al integrar una base de datos de conocimiento de seguridad con herramientas de gestión de eventos e información de seguridad (SIEM), el presente sistema también puede configurarse para actualizar el estado de verificación de las tareas del proyecto y el estado de seguridad de los activos de *software*, tanto durante el desarrollo del *software*, después del lanzamiento como a lo largo de todo el ciclo de vida de la aplicación de *software*. El sistema también puede recibir o consumir retroalimentación durante el tiempo de ejecución o el funcionamiento para poder influir en qué tareas son las más relevantes para la aplicación y repriorizarlas. Los equipos y/o desarrolladores adecuados también pueden responder rápidamente a cualquier retroalimentación o cambio en la lista de tareas priorizadas para cambiar el enfoque y mitigar el riesgo u orquestar el trabajo según las mejores prácticas. La priorización de tareas también puede ajustarse rápidamente basándose en el entorno de seguridad electrónica pública del activo de *software* para dirigir la atención del desarrollador a las cuestiones que requieren una atención más inmediata en cualquier momento durante el SDLC. Una vez que se ha lanzado la aplicación, las actualizaciones a la base de datos de conocimiento pueden desencadenar tareas de acción de actualización del activo de *software* basándose en el contexto de *software* identificado de la aplicación para mantener las normas de seguridad y funcionamiento y mitigar el riesgo del activo de *software* durante su vida útil.

Cada tarea en la lista de tareas priorizadas 114 puede incluir además casos de prueba relevantes personalizados y código de prueba de garantía de calidad de muestra, incluyendo información de seguimiento y registro de auditoría para satisfacer las normas de requisitos y los criterios de auditoría, reglas de mejores prácticas relacionadas con el diseño del código de *software* para producir activos de *software*, así como reglas relacionadas con estructuras de seguridad y procedimientos para la comunicación en Internet y para negocios particulares. Por ejemplo, si el contexto de *software* indica que el activo de *software* se utiliza dentro de una institución financiera que realiza transacciones con tarjetas de crédito, la lista de tareas incluiría regulaciones y marcos de trabajo de control como PCI DSS, COBIT, norma ISO 27001 (anteriormente 17799), GLPA y similares. En otro ejemplo, si el contexto de *software* del proyecto indica que el proyecto se refiere a la industria sanitaria, pueden incluirse regulaciones de privacidad para datos médicos en la lista de tareas. La base de datos de conocimiento 108 puede incluir además reglas de mejores prácticas relativas al diseño de código de *software* para producir activos de *software*, así como reglas relativas a estructuras y procedimientos de seguridad para comunicación en Internet y para negocios particulares.

La combinación de extracción dinámica y continua de datos de contexto para el activo de *software* y una base de datos de conocimiento 108 actualizada periódicamente a lo largo de todo el ciclo de vida del *software* proporciona actualizaciones oportunas a la lista de tareas priorizadas 114 basándose en la información de contexto del activo de *software* y, por tanto, la identificación y solución de tareas y riesgos desde el principio hasta el final durante todo el ciclo de vida del desarrollo del *software*. Sin limitarse a la teoría, uno de los puntos fuertes del sistema y del método descritos actualmente es que los cambios en las bibliotecas de contexto dinámicas, los riesgos recientemente identificados en las bibliotecas estáticas, las nuevas regulaciones de seguridad, las apariciones de *malware* y las actualizaciones de las tareas de seguridad pueden llamar rápidamente la atención de los desarrolladores en cualquier etapa del ciclo de vida del *software* actualizando los requisitos en la base de datos de conocimiento 108. Estas actualizaciones pueden enviarse entonces a la lista de tareas priorizadas 114 para una gestión adecuada de los activos de *software*, opcionalmente con una evaluación de riesgos de seguridad. De esta manera, las tareas actualizadas pueden priorizarse en una lista de tareas priorizadas 114 específicas de aplicación

para el activo de *software*. Por tanto, los desarrolladores pueden recibir alertas rápidas sobre actualizaciones de seguridad relevantes para cada activo de *software* basándose en el contexto de *software* previamente definido y recibir soluciones para evitar riesgos de seguridad en el activo de *software*. En un entorno donde un solo equipo desarrolla y/o mantiene múltiples activos de *software*, también puede informarse al equipo de las actualizaciones de seguridad a medida que se añaden a la base de datos de conocimiento, con alertas de prioridad para requisitos de alta prioridad para responder rápidamente en situaciones de alto riesgo.

La figura 2 ilustra el almacenamiento y la extracción de contexto en el presente sistema. Los repositorios de contexto 202 son bases de datos o sistemas de almacenamiento que contienen información de contexto sobre el proyecto de activos de *software* y comprenden datos de contexto que pertenecen a la construcción y/o el funcionamiento del *software*. Los datos de contexto para un proyecto de *software* pueden almacenarse en el código fuente almacenado en una o más fuentes de código fuente 206. El código fuente 206 incluye código que forma parte del proyecto de *software*. Las epopeyas o historias relacionadas con el activo de *software* pueden almacenarse, por lo menos en parte, como lenguaje natural en una o más especificaciones de *software* 210. La especificación de *software* 210 contiene requisitos funcionales, especificaciones de diseño, diagramas, epopeyas, historias de usuario, contratos de desarrollo de *software* y cualquier otra documentación utilizada para especificar el *software* y/o su funcionamiento interno. La especificación de *software* 210 puede contener descripciones en lenguaje natural de características o conjunto de características de contexto, poblaciones de usuarios, jurisdicción u otros detalles de contexto relevantes para el *software*. La especificación de *software* 210 también puede contener descripciones de trabajo en lenguaje natural, tales como sistemas de tickets, o trabajo nuevo que vendrá o se solicita o recomienda en el futuro, tales como errores encontrados o solicitudes de actualizaciones o mejoras.

Un sistema de registro 208 es un repositorio de información que recopila y realiza un seguimiento de hechos sobre aplicaciones y sistemas, a menudo estructurados como una descripción basada en formularios de uno o más activos de *software*. El sistema de registro 208 puede incluir información sobre tipos, unidades de negocio, descripción basada en hechos de un activo y dónde se implementará el *software*. El sistema de registro 208 puede basarse en *software* comercial listo para utilizar (COTS) o puede construirse a medida. Un tipo de almacén de información de sistema de registro 208 es el sistema de Gestión de Cartera de Proyectos (PPM). Un ejemplo de COTS PPM es "Clarity PPM". El PPM almacena una lista de aplicaciones, equipos, detalles presupuestarios sobre las aplicaciones, descripciones de alto nivel sobre el trabajo planificado o completado para las aplicaciones y otros detalles. El sistema de registro 208 también puede considerarse como una colección de hechos sobre un activo de *software*, en el que los hechos pueden incluir la especificación del servidor físico donde se almacenan los activos de *software*, qué licencias se requieren para utilizar una herramienta de terceros particular y otros datos relacionados con la aplicación creada internamente. El sistema de registro también puede contener información preliminar sobre la aplicación, tal como título, cronograma, equipo, características provisionales, política corporativa y normas organizativas. El presente sistema puede comprender uno o más sistemas de registro 208. Los datos de contexto para el activo de *software* también pueden almacenarse en planes de prueba 211, que son un conjunto de instrucciones para validar comportamientos funcionales o no funcionales del *software*. Los planes de prueba 211 pueden estar en forma de lenguaje natural, en formato legible por ordenador, y pueden incluir, pero sin limitarse a, pruebas unitarias, pruebas de apilamiento completo, pruebas de regresión, pruebas de humo y pruebas de rendimiento.

Los sistemas de gestión 204 son un conjunto de repositorios de datos, cada uno de los cuales proporciona un sistema para gestionar detalles del activo de *software* de forma estructurada. Debido a la naturaleza de cómo se describen y almacenan los datos sobre el activo de *software* en los sistemas de gestión 204, pueden explorarse los sistemas de gestión para extraer el contexto relevante sobre el activo de *software*. Los sistemas de gestión 204 pueden incluir, por ejemplo, uno o más sistemas de gestión de dependencias 212, uno o más repositorios de gestión del ciclo de vida de las aplicaciones (ALM) 214, uno o más repositorios de gestión de código fuente (SCM) 216 y/o uno o más repositorios de gestión de configuración 218.

Un sistema de gestión de dependencias 212 proporciona código externo utilizado por un proyecto y almacena o apunta o, de otro modo, proporciona acceso a bibliotecas utilizadas en el activo de *software*. Las bibliotecas utilizadas en el activo de *software* pueden ser privadas, compradas, públicas, propietarias o una combinación de las mismas. La identidad de un sistema de gestión de dependencias 212 puede extraerse del código fuente 206 para identificar qué bibliotecas se utilizan, buscar qué bibliotecas están vinculadas a esa biblioteca e identificar cualquier vulnerabilidad o problema potencial asociado con cualquier biblioteca vinculada. El metanálisis de una fuente de aplicación identifica un conjunto de dependencias en utilización por una aplicación, un activo o un sistema de *software*, así como las dependencias de esas dependencias. Para cada dependencia, se examina el sistema de gestión de dependencias 212 para determinar o identificar cualquier dependencia implícita. Por ejemplo, si una dependencia explícita A (una dependencia identificada por el metanálisis) depende de las dependencias B y C, entonces si una aplicación/sistema utiliza la dependencia A, el contexto sobre la aplicación/sistema también incluirá las dependencias implícitas B y C. Las dependencias B y C también pueden implicar otras dependencias, que también están contempladas en el presente sistema. Algunos ejemplos de un sistema de gestión de dependencias son PyPI, Maven™ y Sonatype Nexus™. Una vez recopiladas todas las dependencias explícitas e implícitas para el activo de *software*, para cada dependencia definida se recopila el contexto de *software* a partir de un sistema de gestión de dependencias, donde el contexto de *software* es un conjunto de datos de dependencia de contexto C_{dep} .

donde:

$C_{dep} \{c_1, \dots, c_i, \dots, c_w\}$

5 y c_i contiene detalles tales como el nombre de la dependencia, los requisitos técnicos para utilizar la dependencia en una aplicación y la clasificación, donde una o más clasificaciones aportan contexto técnico, de negocio o personalizado adicional a la dependencia, y un ejemplo de c_i es:

Nombre: Controlador de base de datos

10

Requisitos técnicos: Necesita credenciales, Red (tipo = TCP, puerto 3306)

Clasificación: Base de datos

15

Al monitorizar las dependencias de los activos de *software*, el sistema puede identificar actualizaciones a bibliotecas dinámicas vinculadas que plantean un riesgo de seguridad o una vulnerabilidad recientemente identificada introducida por una biblioteca vinculada estática, asignarles una clasificación de riesgo y entregarlas a la lista de tareas en un nivel de prioridad basado en el nivel de riesgo que plantea el problema. El alcance profundo del presente sistema en bases de datos y bibliotecas vinculadas y de dependencia permite que cualquier actualización remota se considere como un riesgo de seguridad y se presente según sea necesario para abordar cualquier riesgo introducido en cualquier etapa durante el ciclo de vida del *software*.

20

25

Un repositorio de gestión del ciclo de vida de las aplicaciones (ALM) 214 gestiona requisitos, pruebas, planes, tareas, errores y problemas durante el desarrollo de *software*. Los repositorios de ALM pueden introducir tanto riesgos a través del código que proporcionan, así como ayudar con la mitigación de riesgos al proporcionar información sobre los riesgos. Las herramientas de ALM también pueden almacenar detalles sobre el trabajo planificado o completado para una aplicación o sistema. El trabajo realizado por una ALM a menudo se organiza en unidades llamadas "temas", donde un tema puede ser de tipo "epopeya", "historia" o "tarea". Los requisitos de trabajo a menudo se almacenan como oraciones en lenguaje natural y otros detalles técnicos, tales como extractos de código fuente, así como enlaces a otros sistemas relevantes para el trabajo. Por ejemplo, un gestor de producto de aplicación puede escribir una historia: "Como usuario, necesito poder actualizar mi nombre y apellido". Cuando se inicia el trabajo en esta historia, un equipo de desarrollo traduciría esta definición en lenguaje natural de una nueva característica a una nueva capacidad en la aplicación o el sistema. La definición de trabajo en lenguaje natural puede manifestarse en actualizaciones de archivos de la aplicación/sistema y, en última instancia, como un cambio de estado o comportamiento cuando se implementa la aplicación actualizada. Puede realizarse entonces un seguimiento de cada cambio en el registro de cambios de la aplicación o del sistema. El trabajo definido en un tema de ALM también puede proporcionar detalles sobre los requisitos de negocio y técnicos del proyecto de *software*, incluyendo cualquiera de flujos de datos específicos, experiencia del usuario y otros detalles relevantes necesarios para implementar un cambio. El contexto capturado a partir de un tema de ALM también puede incluir una clasificación del tema, así como cualquier otra información del trabajo, tal como por ejemplo archivos modificados para implementar el trabajo. Algunos ejemplos específicos de repositorios de ALM incluyen sistemas de seguimiento y emisión de tickets de errores tales como JIRA™ y CA Agile Central.

30

35

40

45

Un repositorio de gestión de código fuente (SCM) 216 realiza un seguimiento del código fuente de la aplicación y del sistema, y es donde se mantiene el código y se realiza un seguimiento de los cambios. El repositorio de SCM 216 puede incluir un registro de cambios detallado que incluye la hora, la descripción del cambio, anotaciones y el nombre del activo de aplicación afectado para el código fuente, así como identificadores de activos de *software* tales como una lista de miembros del equipo, versiones y configuración de construcción. Uno o más repositorios de gestión de código fuente 216, cuando están ubicados externamente, pueden introducir nuevas vulnerabilidades o pueden eliminar las existentes cuando el código que requiere el proyecto de *software* se actualiza por partes externas. Algunos ejemplos específicos de un repositorio de SCM 216 incluyen github.com, bitbucket.org, una base de datos de subversión interna y otros repositorios git. En un ejemplo específico de un repositorio de SCM 216, los datos de contexto pertenecientes al activo de *software* pueden definirse como el conjunto $C_{código}$, donde:

50

55

$C_{código} \{c_1, \dots, c_j, \dots, c_n\}$,

y el elemento de contexto j^o podría incluir detalles tales como:

Fecha – la fecha y la hora en que se realiza el cambio.

60

Usuario – el usuario que realiza el cambio

Descripción – detalles sobre el cambio

65

Archivos – el activo afectado por el cambio

Código – registro de los cambios de código reales

Un repositorio de gestión de configuración 218 puede almacenar información sobre cómo está configurado un sistema y dónde y/o cómo debe implementarse un activo de *software*. Los sistemas de gestión de configuración también almacenan información para soportar la implementación de una aplicación, así como para garantizar un rendimiento y un funcionamiento constantes durante su implementación, tal como, por ejemplo, qué bibliotecas deben instalarse en el sistema, cómo está configurado el servidor web, cómo está configurado el sistema con respecto a, por ejemplo, las reglas de cortafuegos, las reglas de acceso de usuarios y los detalles de acceso a la red. Algunos ejemplos no limitativos de sistemas de gestión de configuración incluyen Puppet™, Ansible™, Chef™ y SaltStack™. Los datos de contexto en por lo menos un repositorio de gestión de configuración 218 pueden describirse como C_{conf} que comprende un conjunto de t elementos de contexto, donde:

$$C_{conf} \{c_1, \dots, c_i, \dots, c_t\},$$

y c_i es un hecho de gestión de configuración sobre una aplicación o sistema, donde c_i puede incluir detalles como una lista de servicios tales como Nginx™, Apache™, MySQL™, detalles de cortafuegos, sistemas operativos, bibliotecas y configuración para servicios y aplicaciones.

Un motor de extracción de contexto 220 extrae el contexto de *software* de los repositorios de contexto 202 utilizando una pluralidad de herramientas, tales como, por ejemplo, un escáner de código 222 y un procesador de lenguaje natural 224. Utilizando un escáner de código 222, puede extraerse el contexto de *software* del código fuente 206, incluyendo una pluralidad de detalles sobre el activo de *software* que pueden incluir, pero sin limitarse a, el lenguaje de programación, la arquitectura del sistema, las ubicaciones de los archivos, los tipos de archivos, las bibliotecas utilizadas y otros detalles. El escáner de código 222 puede ser, por ejemplo, un escáner de pruebas de seguridad de aplicaciones dinámicas (DAST), un escáner de pruebas de seguridad de aplicaciones estáticas (SAST), una herramienta de correlación de vulnerabilidad de aplicaciones, escáneres de agregación, escáneres de pruebas de seguridad de aplicaciones interactivas (IAST), escáneres de protección de seguridad de aplicaciones en tiempo de ejecución (RASP) o una combinación de los mismos. El escáner de código 222 también puede utilizarse para analizar el código escaneado para proporcionar información de contexto adicional para ayudar al módulo de selección a seleccionar los requisitos de seguridad para el activo de *software*. Se utiliza un procesador de lenguaje natural (NLP) 224 para extraer datos de contexto almacenados como lenguaje natural de cualquier repositorio de contexto 202 que contenga lenguaje natural, tal como el repositorio de ALM 214, el sistema de registro 208 o la especificación de *software* 210. Los datos de contexto extraídos se almacenan en una base de datos de contexto 228, que contiene el contexto extraído sobre el activo de *software*.

Un escáner de código se utiliza para escanear código, extraer el contexto de *software* del código fuente 206 y del repositorio de SCM 216, capturar vulnerabilidades de código o detectar deficiencias de código que pueden implicar riesgos en un activo de *software* o en un entorno operativo de *software*. También pueden integrarse en el presente sistema una variedad de escáneres de código 222 tanto para extraer contexto del código fuente como para identificar vulnerabilidades en el código fuente. El escáner de código puede incluir, pero sin limitarse a, escáneres de pruebas de seguridad de aplicaciones estáticas (SAST), escáneres de seguridad de análisis dinámico o de pruebas de seguridad de aplicaciones dinámicas (DAST), escáneres de agregación, escáneres de pruebas de seguridad de aplicaciones interactivas (IAST) y escáneres de protección de seguridad de aplicaciones en tiempo de ejecución (RASP). También pueden incorporarse herramientas de pruebas de seguridad en tiempo de ejecución para generar reglas para pruebas en tiempo de ejecución y actualizar dinámicamente la lista de tareas de seguridad. Los escáneres utilizan heurística y medios automatizados para explorar los comportamientos de un activo de *software* para detectar debilidades. La sincronización avanzada y dinámica con escáneres de código puede proporcionar capacidades adicionales, incluyendo la actualización automática de la lista de tareas de requisitos basándose en la información de vulnerabilidad de la herramienta de escaneo cuando se actualiza el escáner de código, y la actualización automática de las configuraciones de proyecto basándose en la vulnerabilidad o los metadatos o los detalles del contexto detectados por la herramienta de escaneo de código. El conjunto completo de datos de contexto extraídos también se denomina perfil de contexto para el activo de *software* y se almacena en la base de datos de contexto 228.

También puede utilizarse una interfaz de usuario 226 para obtener información adicional para la base de datos de contexto 228 y el motor de extracción de contexto 220 a través de la entrada directa del usuario. La comunicación entre la interfaz de usuario 226 y el motor de extracción de contexto 220 puede ser bidireccional para permitir ciclos de refinamiento por parte del motor de extracción de contexto basándose en la entrada del usuario.

La figura 3 ilustra un método de flujo de datos desde los repositorios de contexto 302 en el presente sistema hasta una base de datos de contexto 318. Los repositorios de contexto 302 contienen datos relacionados con el activo de *software* nuevo o existente que se exploran para extraer el contexto de *software* para crear un perfil de contexto para el activo de *software*. La extracción de contexto 310 puede realizarse mediante una variedad de procesos para extraer datos de contexto dependiendo de cómo se almacenan los datos en cada uno de los repositorios de contexto 302. Los datos de un sistema de registro 304 pueden procesarse o dirigirse a la base de datos de contexto 318 con un procesamiento mínimo o nulo. El código fuente 306 puede procesarse escaneando el código 312

utilizando uno o más escáneres de código. El código fuente 306 también puede procesarse y extraerse contexto realizando un metanálisis 314 de los datos para extraer el significado de las palabras y el código y/o escaneando el código 312 del activo de *software* si el desarrollo ha comenzado y existe código fuente para escanear. La configuración 307 de un activo de *software* incluye su especificación de operación que describe su entorno de ejecución y funcionamiento. Los datos de contexto en una configuración 307 pueden procesarse escaneando la configuración 313 y/o enviando directamente a la base de datos de contexto 318. Un escaneo de configuración 313 revela las configuraciones del activo de *software*, que pueden estar especificadas por un fabricante y personalizadas localmente de manera opcional para adaptarse a las políticas locales. Los datos de contexto de una especificación de *software* 308 pueden contener historias o epopeyas en forma de lenguaje natural, y el lenguaje natural se procesa mediante el procesamiento de lenguaje natural 316, utilizando opcionalmente un sistema de inteligencia artificial. Los datos de contexto en los planes de prueba 309 almacenados como lenguaje natural pueden procesarse mediante procesamiento de lenguaje natural 316, y los datos de contexto en los planes de prueba 309 almacenados como formato legible por ordenador pueden extraerse realizando un metanálisis 314. Todos los datos de contexto extraídos se almacenan entonces en una o más bases de datos de contexto 318.

La figura 4 ilustra un método de identificación y verificación automatizadas de tareas en un ciclo de vida del *software*. Las etapas del método mostradas en la figura 4 pueden realizarse en cualquier momento durante el ciclo de vida del *software* y, preferentemente, a lo largo de todo el ciclo de vida del *software*. El contexto de *software* se obtiene 402 escaneando, raspando o extrayendo datos de contexto de múltiples fuentes relacionadas con el activo de *software*. Los datos de contexto se recopilan de un sistema de registro 404. Se realiza un metanálisis del código de *software* 406 para extraer el contexto del código, así como el lenguaje natural incorporado. Además, las fuentes de datos de especificación de *software* se procesan utilizando procesamiento de lenguaje natural (NLP) 408, opcionalmente con inteligencia artificial, y se recopila información de contexto de uno o más repositorios de contexto 410. Luego se crea o actualiza la base de datos de contexto 414 para el proyecto de *software* que comprende todos los datos de contexto recopilados de múltiples fuentes de datos de contexto. Los requisitos de tarea se seleccionan de una base de datos de conocimiento 420 basándose en el contexto de *software* identificado, donde la selección se realiza de un conjunto de elementos de *software*, tareas o requisitos en la base de datos de conocimiento. En particular, el sistema hace corresponder los datos de contexto de *software* con contenido de la base de datos de conocimiento 422 para seleccionar tareas relevantes para el activo de *software*. Las tareas relevantes se seleccionan basándose en el contexto del activo de *software*; por ejemplo, si la jurisdicción del *software* se limita a Europa, solo se seleccionarán las tareas relevantes para los requisitos y normas europeos, y no aquellas específicas de otras jurisdicciones. Una vez que se han identificado los requisitos para un proyecto de *software*, se genera una lista de tareas priorizadas 424 con unos requisitos de tarea para proporcionar una lista de tareas priorizadas que comprende directrices de desarrollo y requisitos de seguridad relevantes para el proyecto de *software* basándose en el perfil de contexto extraído para el activo de *software*. La priorización de los requisitos de tarea puede basarse en parte en un algoritmo de medición cuantitativa, tal como la base de datos del Sistema de puntuación de vulnerabilidades comunes (CVSS), para indicar el riesgo de una debilidad, confidencialidad o integridad particular, o de vulnerabilidades identificadas por un escáner de código de vulnerabilidad u otro método. Cuando se ha abordado una tarea de la lista de tareas, se utiliza un procedimiento de verificación para verificar la finalización de tarea 426 sometiendo a prueba el *software* de aplicación para garantizar que la tarea se ha abordado adecuadamente, lo que también puede proporcionarse en las directrices de requisitos de seguridad. Las directrices para cada requisito de tarea en la lista de tareas contienen preferentemente una identificación de la tarea que va a abordarse, la solución y un procedimiento de verificación, todo en una sola ubicación para facilitar la implementación, la comprensión y la integración por parte del desarrollador.

La verificación de la finalización de tarea 426 puede realizarse manualmente y/o enviarse a un escáner de código para escanear el código de *software* 412. El escaneo del código 412 también puede identificar la vulnerabilidad del código durante la codificación o verificación en el desarrollo de *software*, así como extraer información de contexto adicional del código del activo de *software*. Al identificar qué vulnerabilidad, información de contexto y códigos de identificación de seguridad, los escáneres pueden extraer e identificar de manera eficaz y cuáles no, los desarrolladores humanos pueden orientarse para que concentren recursos de pruebas humanas más costosos en problemas que los escáneres de códigos automatizados no pueden detectar. También puede proporcionarse una herramienta de prueba de garantía de calidad (QA) que incluya normas de pruebas como casos de prueba en los sistemas de prueba de QA para que el personal de QA pueda realizar un seguimiento de las secuencias de comandos ejecutadas automáticamente y realizar un seguimiento de la cobertura de la prueba y la tasa de aprobación. En un ejemplo, WebInspect™ puede proporcionar normas sobre cómo someter a prueba una debilidad particular o, alternativamente, notificar a un usuario si la prueba de debilidad se realiza automáticamente. También pueden utilizarse una variedad de herramientas de verificación, así como la verificación manual, para confirmar que una tarea o un requisito particular se ha abordado adecuadamente. También pueden utilizarse múltiples escáneres de código para combinar información de verificación de diferentes herramientas de escáner de código en un único resultado de verificación. Si se utilizan múltiples escáneres para escanear el código de *software*, pueden añadirse los resultados de las múltiples herramientas para proporcionar un único resultado de verificación y, opcionalmente, un valor de confianza asociado, siempre que se haya confirmado que cada una de las múltiples herramientas es eficaz para realizar la verificación para la cual el escáner de código indica un resultado de verificación aprobado. El resultado de verificación puede aparecer como un indicador dentro de una vista de tareas de la lista de tareas priorizadas, dentro del requisito de tarea o en diversas ubicaciones en la interfaz gráfica de

usuario del sistema. El sistema también puede identificar tareas de la lista de tareas priorizadas que requieren verificación adicional y repriorizar las tareas en consecuencia. Para cada tarea, el sistema también puede realizar un seguimiento sobre si la tarea requiere pruebas manuales y en qué herramientas de escaneo puede confiarse para verificar el cumplimiento. La verificación puede asignarse de manera manual o automática mediante la retroalimentación del escáner. La integración del escáner de código puede identificar adicionalmente el código asociado con una tarea y proporcionar aprobación/no aprobación basándose en ese escáner, así como volver a verificar dinámicamente el código a lo largo de todo el SDLC para presentar cualquier tarea de seguridad que haya fallado en la verificación posterior. Pueden recopilarse otros datos de contexto, tal como qué desarrollador resolvió una tarea particular, cuánto tiempo le llevó completarla y la hora y fecha de verificación, lo que proporciona además seguimiento y responsabilidad por la realización de actividades de seguridad del *software*. La identificación de que se siguieron normas individuales proporciona trazabilidad y permite a las partes interesadas del SDLC comunicar su norma de finalización para cada fase.

La lista de tareas de requisitos puede actualizarse con el estado de las tareas para verificar la finalización de tarea 426 basándose en el estado de verificación de cada una de las tareas enumeradas en la lista de tareas priorizadas. En un ejemplo, una tarea puede marcarse con un estado de verificación de "Por realizar" o "Realizado" basándose en los resultados de la verificación o la forma en que se abordó la tarea, por ejemplo, "máquina" o "humano". Si una tarea se ha verificado suficientemente, entonces la tarea puede marcarse como "Realizada". En los casos en que se necesite una verificación adicional, la tarea puede permanecer marcada como "Por realizar" de manera que pueda aplicarse un esfuerzo adicional para verificar que la tarea se haya abordado correctamente. En otro ejemplo, un estado de verificación puede ser múltiple y puede asignarse uno de entre una pluralidad de niveles de estado, tales como "Verificación aprobada", "Verificación fallida", "Verificación parcialmente aprobada" o "Aprobado", con detalles adicionales sobre cómo se asignó el nivel de estado de verificación, tal como si se verificó por una máquina o por un ser humano. Una tarea con una "Verificación aprobada" puede evaluarse adicionalmente como de bajo riesgo y abordarse adecuadamente. Una tarea marcada como "Verificación fallida" puede indicar que se identificó una vulnerabilidad, que una tarea no se completó correctamente o que debe abordarse y que se debe trabajar en ella o examinarla nuevamente. Por tanto, a la tarea se le asigna un estado incompleto. Una tarea marcada con una "Verificación parcialmente aprobada" puede indicar que sus controles e instrucciones de mitigación necesitan un esfuerzo adicional para confirmarla como verificada. Una tarea también puede marcarse con un "Estado sin verificación" cuando no hay ningún estado de verificación disponible. En este caso, o bien se sabe que la máquina utilizada para abordar la tarea es incapaz o poco fiable para completar la tarea, o bien se desconoce la capacidad de la máquina encargada de la tarea para identificar si una tarea particular se ha abordado correctamente. Una verificación de "Estado sin verificación" también puede indicar que los controles e instrucciones asociados con la tarea necesitan trabajo adicional para confirmarse como verificados. Pueden proporcionar detalles de los resultados de la verificación al desarrollador para orientar esfuerzos futuros. Tras la verificación de la finalización de tarea 426, el sistema puede registrar la finalización de tarea 428, incluyendo contexto adicional sobre quién/qué completó la tarea, qué se hizo para verificar la tarea y cuándo se completó la tarea, lo que contribuye a contexto adicional para la base de datos de contexto. La finalización de una tarea también puede registrarse 428 para proporcionar evidencia de auditoría tangible y rastrear el cumplimiento de las normas de seguridad, cumplimiento o regulatorias. Una vez que se ha completado un requisito o una tarea, puede designarse como tal haciendo clic manualmente en un botón o introduciendo datos, dando a los equipos del proyecto la posibilidad de auditar qué usuario y en qué momento completó qué acciones. Los gestores de proyectos pueden realizar un seguimiento del registro de finalización de tareas para confirmar que los desarrolladores han seguido las directrices de programación y las normas de desarrollo seguros. La lista de tareas también permite que múltiples usuarios actualicen el sistema y completen tareas simultáneamente y proporciona la capacidad de paralizar el cambio de las propiedades del proyecto. El cumplimiento a nivel empresarial de la legislación y/o regulación relevante sobre seguridad de aplicaciones puede actualizarse basándose en la finalización de la tarea y puede mostrar el estado de cumplimiento con, por ejemplo, PCI DSS, N ERC CIP, GLBA e HIPAA para el proyecto. Pueden generarse y realizarse el seguimiento de directrices de auditoría y regulatorias específicas para su utilización posterior para generar los informes requeridos, y pueden recopilarse datos adicionales sobre cuándo se emplean, implementan y prueban las directrices, y también pueden generarse informes de cumplimiento para fines de auditoría, así como para fines regulatorios, tal como demostrar el cumplimiento de los requisitos, la implementación y las directrices de prueba. También puede utilizarse una puerta de liberación 430 para la validación de calidad o la verificación de la finalización de una tarea.

En la creación o actualización de la base de datos de contexto 414, puede obtenerse una mayor aclaración obteniendo la entrada de usuario 416 para proporcionar un contexto adicional para actualizar la base de datos de contexto, lo que podría incluir actualizar o añadir contexto relevante, aclarar el contexto o eliminar contexto irrelevante. En particular, además del contexto de *software* extraído automáticamente, puede obtenerse entrada adicional de los usuarios 416, en la que el sistema puede solicitar a un desarrollador que proporcione información adicional sobre el contexto de *software* para seleccionar mejor los requisitos de tarea apropiados a partir de la base de datos de conocimiento y para aumentar el perfil de contexto para el activo de *software* en la base de datos de contexto. La consulta para la entrada de los usuarios 416 puede proporcionarse como una lista de consultas de contexto de *software* seleccionadas a partir de las cuales el desarrollador puede identificar detalles sobre el contexto del activo de *software* para el cual el sistema no puede extraer o requiere verificación. El aprendizaje automático también puede comparar perfiles de contexto de un conjunto de prueba de activos de *software* para

saber qué requisitos de tarea son relevantes para un contexto particular identificado para seleccionar tareas con mayor precisión. El sistema también puede dirigir preguntas a los desarrolladores para obtener un perfil más completo del contexto de *software* donde faltan datos, proporcionando enseñanza adicional a una inteligencia artificial sobre las relaciones de contexto para tareas particulares. Las preguntas pueden incluir reglas de inclusión para que el sistema solo haga preguntas relevantes, por ejemplo, preguntar qué versión de Java está utilizándose solo después de que se identifique que el proyecto de *software* utiliza Java. El contexto de *software* identificado permite de ese modo que el módulo seleccione refinar un conjunto en cascada de preguntas más detalladas, así como recuperar directrices y requisitos de seguridad relevantes para desarrollar una aplicación de *software* segura. En un ejemplo, al identificar el lenguaje “perfiles Java EE” de una lista que incluye Java, Tomcat y Apache, se presenta un conjunto de atributos previamente rellenos pertinentes a los perfiles de Java EE para el proyecto. Los atributos de contexto del proyecto pueden detallarse adicionalmente y estar accesibles para las partes interesadas; por ejemplo, puede identificarse que Apache Struts se estaba utilizando en el activo de *software* y puede proporcionarse una orientación específica para Struts que apoye a Struts desde la base de datos de conocimiento a través del módulo de selección. Los desarrolladores también pueden proporcionar detalles adicionales sobre el contexto y las funciones del proyecto, por ejemplo, especificando si la aplicación que está desarrollándose implica interacciones con un sistema operativo, la función de carga de archivos, la autenticación de usuarios finales, etc. En otro ejemplo, una característica del *software* puede ser “la aplicación utiliza Java” y otra podría ser “la aplicación debe cumplir con las normas de seguridad de datos de la industria de tarjetas de pago”, y se seleccionará un requisito de seguridad basándose en la intersección de la combinación de información de contexto.

La figura 5 ilustra un método para extraer datos de contexto para un proyecto y seleccionar requisitos basándose en el contexto extraído. Se configura una base de datos de contexto 502 para un activo de *software* de proyecto nuevo o existente y el presente método genera un conjunto de tareas de trabajo incorporando las configuraciones de proyecto y el contexto de *software* del activo de *software*. A partir de la base de datos de contexto 502 se crea un enlace al proyecto 504, que incluye una ubicación donde pueden actualizarse las configuraciones de proyecto 506, y una ubicación donde puede actualizarse el contexto de *software* 508. Las configuraciones de proyecto pueden incluir detalles de cómo se clasifica un proyecto entre otros proyectos en el sistema, y tanto los participantes humanos como las máquinas pueden contribuir al proyecto, lo que proporciona atributos adicionales que describen el proyecto. En un ejemplo, una base de datos de contexto contiene contexto extraído sobre el activo de *software*, donde el conjunto de datos de contexto para el activo de *software* se define como:

$$C_{\text{contexto}} = \{C_1, \dots, C_j, C_{j+1}, \dots, C_{j+k}, C_{j+k+1}, \dots, C_{j+k+m}, \dots, C_n\}, \text{ donde } 1 \leq j \leq k \leq m \leq n$$

Las tareas se seleccionan de la base de datos de conocimiento 510 basándose en el contexto de *software*. El sistema utiliza los datos de contexto de la base de datos de contexto para crear una lista de requisitos de tarea para el proyecto. Considérese el conjunto de datos de contexto para el activo de *software* C_{contexto} anterior, en el que:

$C_1 = \{C_j, C_{j+1}, C_{j+2}, C_{j+3}, C_{j+4}\}$ indica un subconjunto de C_{contexto} , donde C_1 se compone de una lista de elementos de contexto o datos de contexto discretos utilizados para derivar las configuraciones de proyecto, y

$C_1 =$
 {La aplicación está asignada a la unidad de negocio “Corporativo”,
 El contacto del gestor de proyecto de la aplicación es jsmith@example.com,
 La aplicación se identifica mediante el ID interno “CORP_ID_ABC_123”,
 El “Equipo XYZ” tiene acceso al código fuente de la aplicación en el repositorio de código fuente,
 La herramienta “Fortify” está configurada para escanear la aplicación en busca de debilidades de *software*}

El sistema procesa C_1 y realiza las siguientes operaciones correspondientes a los elementos anteriores.

1. Se crea el proyecto P y se asigna a la unidad de negocio “Corporativo” en el sistema
2. El proyecto P se asigna al usuario indicado por “jsmith@example.com”
3. Al Proyecto P se le asigna un identificador “CORP_ID_ABC_123”
4. El grupo identificado como “Equipo XYZ” se asigna al Proyecto P.
5. Se configura una integración de escáner Fortify para P

El contexto de *software* se calcula entonces a partir de los datos de contexto en la base de datos de contexto. Considérese un segundo subconjunto C_2 de datos de contexto donde:

$C_2 = \{C_{j+k+1}, \dots, C_{j+k+m}\}$ de C_{contexto} , que presenta m elementos de contexto.

El sistema computa entonces un contexto de *software* para el Proyecto P, donde el contexto de *software* comprende las características del entorno de *software* que pertenecen a la codificación y ejecución del activo de *software*. Luego se realiza la correspondencia de los datos de contexto de *software* con los requisitos de una base de datos de conocimiento 512 haciendo corresponder el perfil de contexto y los datos de contexto individuales en la base de datos de contexto para el activo de *software* con los requisitos etiquetados correspondientes en la base de datos de conocimiento. Un bucle de retroalimentación de aprendizaje 518 puede monitorizar además patrones de extracción, aumento y adición de contexto por parte de los desarrolladores para crear un modelo de perfiles de contexto utilizando aprendizaje automático y/o inteligencia artificial. Es posible que puedan encontrarse nuevas tareas durante la verificación o el registro de la finalización de tareas que no se capturaron inicialmente durante la extracción de contexto. El bucle de retroalimentación de aprendizaje puede luego actualizar el proceso de selección de tareas para que estas tareas se capturen en proyectos futuros. Por ejemplo, si la lista de tareas priorizadas excluye una tarea que un desarrollador añade al proyecto, el sistema puede añadir automáticamente la tarea al perfil de contexto de un próximo proyecto que presente un perfil de contexto similar. De la misma manera, si se incluye un requisito particular en la lista de tareas para un proyecto y un desarrollador considera que la tarea no es relevante para un proyecto en particular, el desarrollador puede eliminar la tarea y el sistema puede aprender a no incluir esa tarea en proyectos futuros con perfiles de contexto similares. De esta manera, el modelo de aprendizaje automático puede actualizarse cuando se añade o elimina una tarea de la lista de tareas, o cuando se actualizan las reglas de correspondencia que hacen corresponder contexto extraído con las tareas en la base de datos de conocimiento. En un ejemplo, un proceso humano o de máquina puede identificar además que una tarea de trabajo T1 del tipo T identificada por el sistema no es relevante para el Proyecto P. El sistema incorpora esta retroalimentación y actualiza las reglas de correspondencia internas para la tarea de trabajo T1 de modo que en el futuro, T1 no se incluya en un proyecto que presente un contexto de *software* similar. Alternativamente, un proceso humano o de máquina puede incluir una tarea de trabajo T2 en un Proyecto P. El sistema puede entonces actualizar las reglas de correspondencia internas para que T2 se incluya en proyectos que presenten perfiles de contexto de *software* similares.

La figura 6 ilustra un método para orquestar el flujo de trabajo para sistemas humanos o automatizados. Los requisitos de tarea que deben abordarse se presentan en la lista de tareas priorizadas en orden de prioridad dependiendo de la naturaleza de la tarea. Pueden estar involucrados procesos humanos o de máquina para implementar tareas seleccionadas de la base de datos de conocimiento dirigiendo automáticamente los requisitos abordables por máquina a los activos informáticos capaces de abordar el requisito, reduciendo así la necesidad de intervención humana. Una vez que se identifican las tareas y se ha creado la lista de tareas priorizadas para un proyecto, el sistema evalúa cada tarea en la lista de tareas priorizadas para determinar si la tarea puede abordarse de manera razonable por un activo de máquina 602, y separa las tareas abordables por máquina de aquellas que requieren intervención humana 604. Las tareas particulares en la base de datos de conocimiento pueden categorizarse como que requieren verificación manual o humana basándose en el conocimiento de que los requisitos asociados con estos elementos no pueden detectarse, mitigarse y/o someterse a prueba de manera adecuada por métodos automatizados, y dirige estas tareas que no pueden abordarse de manera fácil o fiable mediante un proceso automático o por máquina, o que requieren verificación humana, a la lista de tareas priorizadas de un desarrollador humano 608. La selección de tareas en la lista de tareas priorizadas que requieren intervención manual puede realizarse utilizando un filtro de tareas para identificar cuáles tareas son las más adecuadas y/o requieren intervención humana para un análisis y verificación adicionales, y qué tareas pueden abordarse de manera adecuada por una máquina. Las tareas abordadas de manera razonable o fiable por un activo informático automático o de máquina se envían a la lista de tareas priorizadas de un activo informático 606 para abordarse. Los resultados de la lista priorizada por la máquina o el desarrollador pueden clasificarse 610 y evaluarse entonces por un ser humano o una máquina para analizar el resultado de la lista de tareas abordables por máquina tal como se generó en la etapa de separación de las tareas abordables por máquina de las que requieren intervención humana 604. Los resultados falsos positivos se identifican en la clasificación 610 de lista y se eliminan y vuelven a evaluarse basándose en la experiencia (desarrollador) o el conocimiento del contexto (máquina). Un bucle de retroalimentación puede dirigir cualquier tarea dada de regreso a la lista priorizada de tareas abordables por máquina a los activos informáticos 606 y puede reasignarse una máquina para verificar la misma tarea, pero con nuevas instrucciones. Alternativamente, cualquier tarea clasificada puede trasladarse de la lista de tareas abordables por máquina a la lista de tareas que requieren intervención humana para desarrolladores humanos, según lo clasificado y determinado por una máquina o un desarrollador. En la lista de tareas priorizadas de un desarrollador humano 608, un ser humano puede identificar una tarea como la que se verifica mejor mediante una máquina, y el sistema y/o la base de datos de conocimiento que contiene detalles sobre la tarea pueden actualizarse con una nueva etiqueta para garantizar que futuras tareas similares se dirijan a una máquina para su procesamiento 606, y viceversa. Si se produce un error durante el proceso o si los detalles indican que la máquina no puede validar la tarea, entonces el sistema se actualiza con información sobre la capacidad del activo de máquina o la falta de la misma para abordar la tarea, y la tarea se redirige a la lista de tareas priorizadas de un desarrollador humano 608 para su reprocesamiento. La verificación de la finalización de tarea 612 para cada tarea en la lista de tareas priorizadas combinadas para el proyecto puede manejarse de manera similar, evaluando si la verificación de cada tarea puede abordarse razonablemente por un activo informático o requiere verificación humana. Cada tarea en la base de datos de conocimiento puede incluir detalles sobre qué tareas requieren verificación humana, y puede acoplarse con el motor de priorización para proporcionar una lista de tareas priorizadas y asignarle una puntuación de confianza de verificación para que los desarrolladores lo aborden, lo que

optimiza la utilización del valioso tiempo del desarrollador y guía la atención del desarrollador a la solución de tareas de seguridad de mayor riesgo que no están bien identificadas por las herramientas automatizadas. Una vez que la tarea se completa y se verifica, se elimina de la lista de tareas priorizadas para el proyecto. Al priorizar y orquestar tareas entre sistemas automatizados y desarrolladores humanos, las tareas de desarrollo de *software* y seguridad pueden agruparse en un conjunto de tareas automatizadas y un conjunto de tareas manuales, de modo que las tareas se aborden de manera conveniente en orden de prioridad y se controle el riesgo.

La figura 7 ilustra un flujo de trabajo para incorporar uno o más proyectos. En particular, el sistema descrito actualmente puede descubrir automáticamente activos de *software* en una organización e incorporar uno o más activos de *software* al sistema para rastrear el activo de *software* a lo largo de todo su ciclo de vida. En la incorporación de uno o más proyectos nuevos o existentes, se seleccionan e identifican el proyecto o proyectos que va(n) a incorporarse 702. La incorporación del proyecto 702 puede basarse en el número o el contenido de los tickets, por ejemplo, en una base de datos épica, o por prioridad de negocio de la organización, facilidad de resolución, especificación del *software* o por evaluación de riesgos. Se extrae entonces el contexto disponible sobre cada proyecto 704 mediante uno o más motores de extracción. El contexto de archivo sobre un proyecto puede identificarse utilizando, por ejemplo, la secuencia de comandos `<scan.py--dir/project/path/--operation=analyze>` Algunas características que pueden extraerse sobre un proyecto desde su directorio fuente, por ejemplo, son los tipos de archivos ("característica:tipo de archivo:java") y los nombres de archivos ("característica:archivo:AndroidManifest.xml"). Luego se analiza el contexto de proyecto 706 y se utiliza información sobre los archivos de proyecto y los tipos de archivos para determinar determinados detalles para mapear el contexto de proyecto en las tareas en la base de datos de conocimiento. En algunos ejemplos:

SI archivo = "Rakefile" y hay otros archivos .rb asociados ENTONCES proyecto Ruby

SI archivo = "AndroidManifest.xml" entonces proyecto Android

SI hay muchos archivos .java ENTONCES proyecto Java

SI hay muchos archivos .java y web.xml ENTONCES proyecto web Java

En un ejemplo, las dependencias del proyecto se examinan automáticamente para buscar cualquier pista sobre el contexto de proyecto. Los archivos de proyecto pueden incluir, por ejemplo, dependencias, detalles de sistemas internos, mensajes de confirmación de código, hallazgos del escáner de código, epopeyas, historias y errores, todos los cuales pueden considerarse como contexto de *software* y contribuir con datos de contexto al conjunto de contexto del activo de *software*. En un ejemplo, puede utilizarse la secuencia de comandos `<scan.py--dir/project/path/--operation=analyze--dep=dependency-check-report.json>` para encontrar las dependencias:

```
dep:stapler
dep:http2-common
dep:jetty-webapp
dep:guava
dep:commons-beanutils
```

En otro ejemplo, un metanálisis puede extraer significado de las palabras, tal como examinando los mensajes de confirmación en Git de un proyecto para cualquier tema, o examinando un sistema de tickets para temas en una epopeya, un conjunto de historias o tareas, dependencias, detalles de sistemas internos, mensajes de confirmación de código, hallazgos del escáner o errores. Por ejemplo, puede utilizarse la secuencia de comandos `<scan.py --dir/project/path/--operation=analyze --tool_config=tool_config.json>` para encontrar temas:

```
tema: PCI
tema: PII
tema: utiliza_contraseña
tema: utiliza_base de datos
```

También pueden realizarse predicciones de contexto de *software* adicional utilizando el contexto de *software* identificado. Por ejemplo, puede examinarse un directorio de fuente de proyecto para determinar qué tareas de la base de datos de conocimiento se adaptan al perfil de contexto utilizando la secuencia de comandos `<scan.py--dir/project/path/--operation=predict--train_path=training/*.json>` para encontrar tareas de conocimiento relevantes. Puede devolverse una lista de resultados de ejemplo, tal como se muestra en la tabla 2:

Tabla 2

Predicción	Confianza
respuesta:php	0.00

perfil:Android	0.75
respuesta:ruby	0.00
respuesta:python	0.00
respuesta:Java	0.91

El sistema también puede entrenarse para detectar nuevos proyectos. Un ejemplo de archivo de entrenamiento para proyectos PHP podría ser:

```

5      [
      {
        "características": [
          "archivo:composer.json",
          "tipo de archivo:.php"
10      ],
        "etiquetas": [
          "respuesta:php",
          "perfil:php"
15      ]
      }
    ]

```

De este modo, el sistema puede aprender de los proyectos existentes, por ejemplo, realizando consultas utilizando la secuencia de comandos `<scan.py--dir/project/path/--operation=learn--train_path=new_training.csv--project_id=2467>`. En otro ejemplo, el sistema puede extraer un perfil de requisitos o un conjunto de requisitos relacionados con el contexto y los requisitos coincidentes de la base de datos de conocimiento para un proyecto existente. Puede extraerse el contexto del proyecto existente de su archivo, identificar las dependencias y otras fuentes, y combinar los dos conjuntos de información para crear un conjunto de entrenamiento para futuras predicciones de los requisitos de proyecto para proyectos con perfiles de contexto similares. El sistema puede crear así automáticamente proyectos utilizando el contexto `<scan.py--dir/project/path/--operation=create_project--trust=0.40--train_path=training/*.json>`

Una vez creado el proyecto, el proyecto se importa 708 y luego se establece el perfil del proyecto 710. Pueden especificarse detalles adicionales durante la incorporación, tales como resultados adicionales de encuestas de proyectos, asignación de membresía de proyectos e integraciones configuradas.

El presente sistema puede integrarse con otras herramientas de gestión del ciclo de vida de las aplicaciones (ALM) que proporcionan un sistema de tickets de trabajo para describir y priorizar el trabajo de los desarrolladores. En una realización, la lista de tareas de requisitos también puede exportarse como un único documento de orientación de desarrollo de *software* o una herramienta de ALM, tal como Atlassian JIRA™. En una ALM, la lista de tareas de requisitos también puede sincronizarse con la herramienta de ALM para permitir que las partes interesadas en seguridad introduzcan requisitos dentro de la herramienta de ALM en el flujo de trabajo de un desarrollador con priorización, mejores prácticas, orientación y análisis de riesgos necesarios para que un desarrollador los implemente. El desarrollador puede continuar trabajando dentro de la herramienta de ALM y, a medida que se completa el trabajo, el presente sistema se puede mantener actualizado con el estado del trabajo y los requisitos correspondientes. Por tanto, una sincronización bidireccional entre el presente sistema y una herramienta de ALM permite a los desarrolladores comunicar la prioridad y el trabajo de seguridad a los miembros del equipo en el sistema ya implementado en un sitio del cliente. También puede utilizarse una interfaz de programación de aplicaciones (API) para construir una aplicación de cliente personalizada que proporcione orientación dirigida y requisitos particulares para una cartera de proyectos del cliente. Una API puede dar acceso programático externo a los datos y comportamientos del sistema, de modo que se puedan realizar consultas e instrucciones al sistema y se le pueda presentar al usuario una lista de tareas actualizada. El desarrollador también puede exportar la lista de tareas de requisitos de seguridad como un documento electrónico estático según sea necesario.

Aunque la presente divulgación se centra en un sistema y un método para la automatización de la identificación de tareas para un único activo de *software*, se entiende que una organización presentará una pluralidad de activos de *software* que se beneficiarían de los mismos. Por consiguiente, múltiples activos de *software* dentro de una organización pueden someterse a identificación de tareas según el sistema y método descritos actualmente, independientemente del estado de desarrollo de cada activo en el ciclo de vida del *software*, proporcionando a la organización una vista y una lista de tareas priorizadas para cada activo de *software*. Además, se entenderá completamente que la organización también podría fusionar las listas de tareas priorizadas para dos o más activos de *software* e incluir potencialmente todos los activos de *software* en la organización, para priorizar el trabajo de desarrolladores individuales, equipos de desarrolladores o de la organización en su conjunto. Hay una multitud de maneras de priorizar adicionalmente una o más listas de tareas fusionadas o globales incluyendo, pero sin limitarse a, la prioridad de la tarea, la prioridad del activo de *software*, la prioridad del proyecto o la prioridad del riesgo. Esta

lista de tareas fusionada también podría actualizarse dinámicamente basándose en el riesgo de un activo de *software* individual u otra prioridad de la organización.

REIVINDICACIONES

1. Método para la automatización de la identificación de tareas para un activo de *software* que va a ser realizado por un sistema informático, comprendiendo el método:

en un procesador, extraer datos de contexto de un repositorio de contexto (102) del activo de *software* realizando un procesamiento de lenguaje natural en por lo menos un repositorio de contexto (102), comprendiendo los datos de contexto unas propiedades técnicas y operativas, y unas características del entorno de *software* que pertenecen a la descripción, codificación y ejecución del activo de *software* y una descripción en lenguaje natural de las características y componentes deseados del activo de *software*, siendo la extracción de datos de contexto dinámica y continua;

en un procesador, realizar un metanálisis de los datos de contexto para generar información sobre el activo de *software* incluyendo la estructura de fuente del sistema de activo de *software*, la estructura de archivos, y un conjunto de dependencias incluyendo bibliotecas en utilización por el activo de *software*;

en un procesador, hacer corresponder los datos de contexto extraídos con unos requisitos de tarea relevantes en una base de datos de conocimiento (108) que comprende una pluralidad de requisitos de tarea accesibles con una herramienta de recuperación haciendo corresponder los datos de contexto para el activo de *software* con unos requisitos etiquetados correspondientes en la base de datos de conocimiento (108), comprendiendo cada requisito de tarea unas directrices de tarea y una orientación práctica para orquestar el desarrollo de *software* y una o más etiquetas que delinear su relevancia para un contexto de *software* particular;

en un procesador, seleccionar unos requisitos de tarea correspondientes específicos para el activo de *software* basándose en la correspondencia de los datos de contexto extraídos;

en un procesador, generar una lista de tareas priorizadas (114) que comprende los requisitos de tarea correspondientes para el activo de *software*, basándose la priorización en la orquestación de tareas o el riesgo de cada uno de los requisitos de tarea seleccionados;

en un procesador, actualizar la lista de tareas priorizadas (114) a lo largo de todo el ciclo de vida del *software* cuando se realizan actualizaciones en el repositorio de contexto dinámico o la base de datos de conocimiento (108) que son relevantes para el activo de *software*, en el que los requisitos de tarea seleccionados de la base de datos de conocimiento (108) se priorizan basándose en uno o más de entre una orquestación de tareas y un riesgo de cada uno de los requisitos de tarea seleccionados, en el que las tareas que se identifican como susceptibles de riesgo se escalan automáticamente en la lista de tareas priorizadas, en el que los requisitos de tarea son requisitos de tarea de seguridad; y

presentar la lista de tareas priorizadas (114) en una interfaz de usuario.

2. Método según la reivindicación 1, en el que los datos de contexto comprenden código fuente y lenguaje natural.

3. Método según una cualquiera de las reivindicaciones 1 o 2, en el que el repositorio de contexto es por lo menos uno de entre un sistema de registro, repositorio de código fuente, repositorio épico, sistema de gestión, sistema de gestión de dependencias, repositorio de gestión de código fuente, repositorio de gestión del ciclo de vida de las aplicaciones y repositorio de gestión de configuración.

4. Método según una cualquiera de las reivindicaciones 1 a 3, que comprende además escanear una pluralidad de repositorios de contexto.

5. Método según una cualquiera de las reivindicaciones 1 a 4, en el que realizar un metanálisis en el código fuente del activo de *software* comprende identificar cualquier vulnerabilidad o problema potencial asociado con cualquier biblioteca vinculada.

6. Método según una cualquiera de las reivindicaciones 1 a 5, que comprende además escanear el código fuente del activo de *software* utilizando un escáner de código para identificar cualquier vulnerabilidad de código.

7. Método según una cualquiera de las reivindicaciones 1 a 6, en el que el método se realiza mediante más de un dispositivo informático.

8. Método según una cualquiera de las reivindicaciones 1 a 7, en el que se extraen datos de contexto del repositorio de contexto con procesamiento de lenguaje natural del repositorio de contexto.

9. Método según una cualquiera de las reivindicaciones 1 a 8, que comprende además calcular el coste de un proyecto de desarrollo de *software* calculando los recursos requeridos para abordar cada tarea en la lista de tareas priorizadas.

10. Sistema para la automatización de la identificación de tareas en un ciclo de vida del activo de *software* que comprende:

- 5 un repositorio de contexto (102) que comprende datos de contexto para el activo de *software* y una descripción en lenguaje natural de las características y componentes deseados del activo de *software*;
- 10 un motor de extracción de contexto (104) en un procesador que comprende instrucciones legibles por ordenador almacenadas en un dispositivo de memoria para extraer los datos de contexto del repositorio de contexto mediante la extracción de los datos de contexto utilizando procesamiento de lenguaje natural, comprendiendo los datos de contexto unos datos que contribuyen a las propiedades técnicas y operativas, y unas características del entorno de *software* que pertenecen a la descripción, codificación y ejecución del activo de *software*, transformando el motor de extracción de contexto los datos de contexto en información utilizable por máquina para el activo de *software*, en el que se realiza un metanálisis de los datos de contexto para generar información sobre el activo de *software* incluyendo la estructura de fuente del sistema de activo de *software*, estructura de archivos, y un conjunto de dependencias incluyendo bibliotecas en utilización por el activo de *software*;
- 20 una base de datos de conocimiento (108) que comprende una pluralidad de requisitos de tarea que comprenden unas directrices de tarea y una orientación práctica para orquestar el desarrollo de *software* accesible con una herramienta de recuperación para almacenar requisitos de tarea y una o más etiquetas que delimitan su relevancia para un contexto de *software* particular;
- 25 un módulo de selección (110) para seleccionar requisitos de tarea de la base de datos de conocimiento relevantes para el activo de *software* haciendo corresponder los datos de contexto extraídos para el activo de *software* con unos requisitos etiquetados correspondientes en la base de datos de conocimiento (108) basándose en los datos de contexto específicos para el activo de *software* para crear una lista de tareas priorizadas (114), actualizando el módulo de selección (110) la lista de tareas priorizadas (114) a lo largo de todo el ciclo de vida del *software* cuando se realizan actualizaciones en el repositorio de contexto dinámico o la base de datos de conocimiento (108) que son relevantes para el activo de *software*; y
- 30 un motor de priorización (112), en un procesador que comprende unas instrucciones legibles por ordenador en la memoria, para priorizar los requisitos de tarea seleccionados de la base de datos de conocimiento (108) basándose en uno o más de entre una orquestación de tareas y un riesgo de cada uno de los requisitos de tarea seleccionados y actualizar la lista de tareas priorizadas (114), en el que las tareas que se identifican como susceptibles de riesgo se escalan automáticamente en la lista de tareas priorizadas, en el que los requisitos de tarea son requisitos de tarea de seguridad;
- 35 una interfaz de usuario para presentar la lista de tareas priorizadas (114).
- 40 11. Sistema según la reivindicación 10, en el que el motor de extracción de contexto comprende un escáner de código y/o un procesador de lenguaje natural.
- 45 12. Sistema según una cualquiera de las reivindicaciones 10 u 11, en el que el repositorio de contexto es un sistema de gestión, preferentemente un sistema de gestión de dependencias, un repositorio de gestión de código fuente, un repositorio de gestión del ciclo de vida de las aplicaciones, y un repositorio de gestión de configuración.

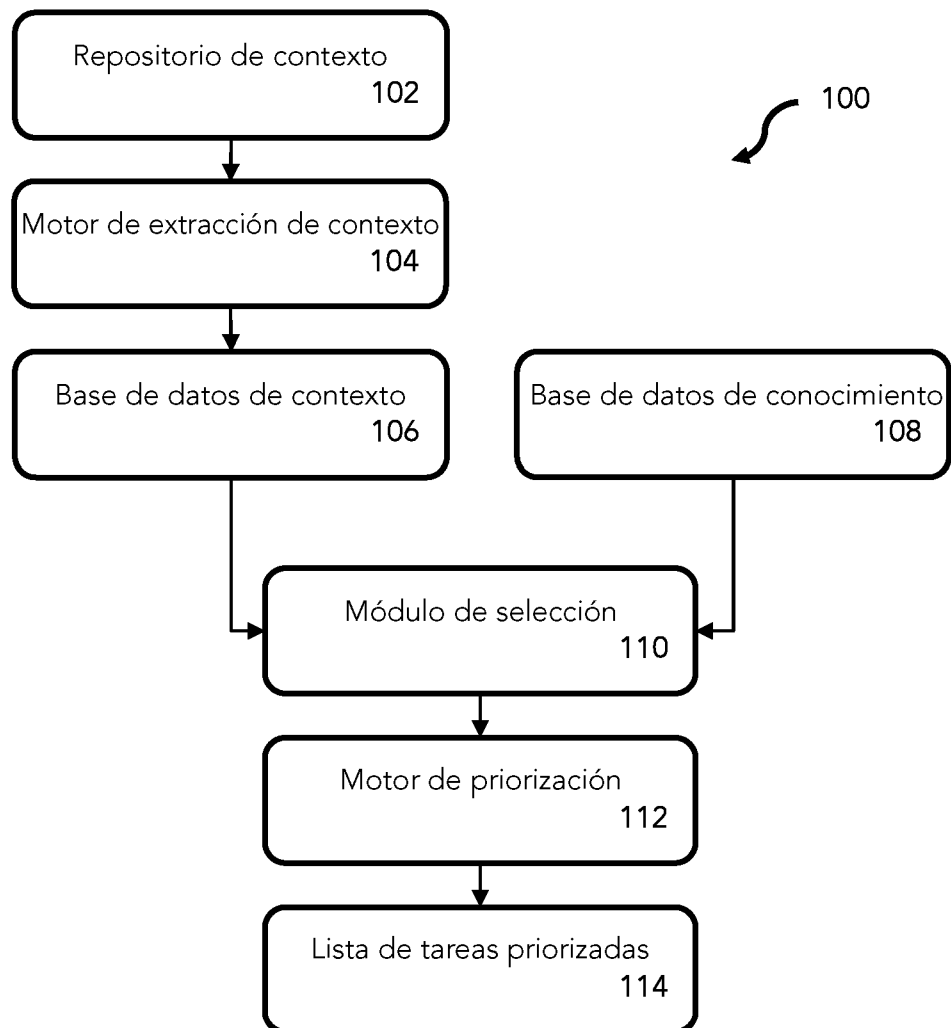


Figura 1

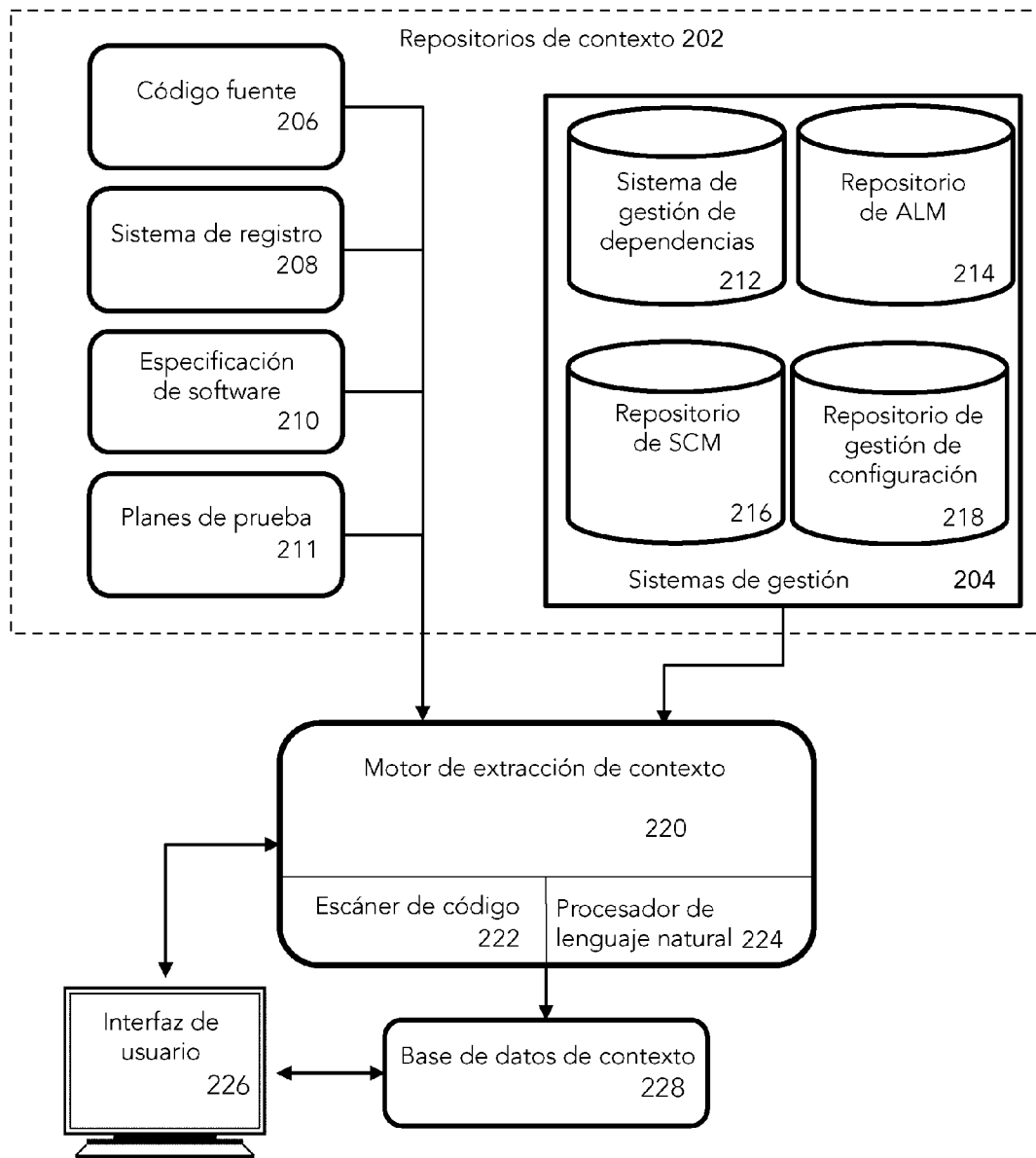


Figura 2

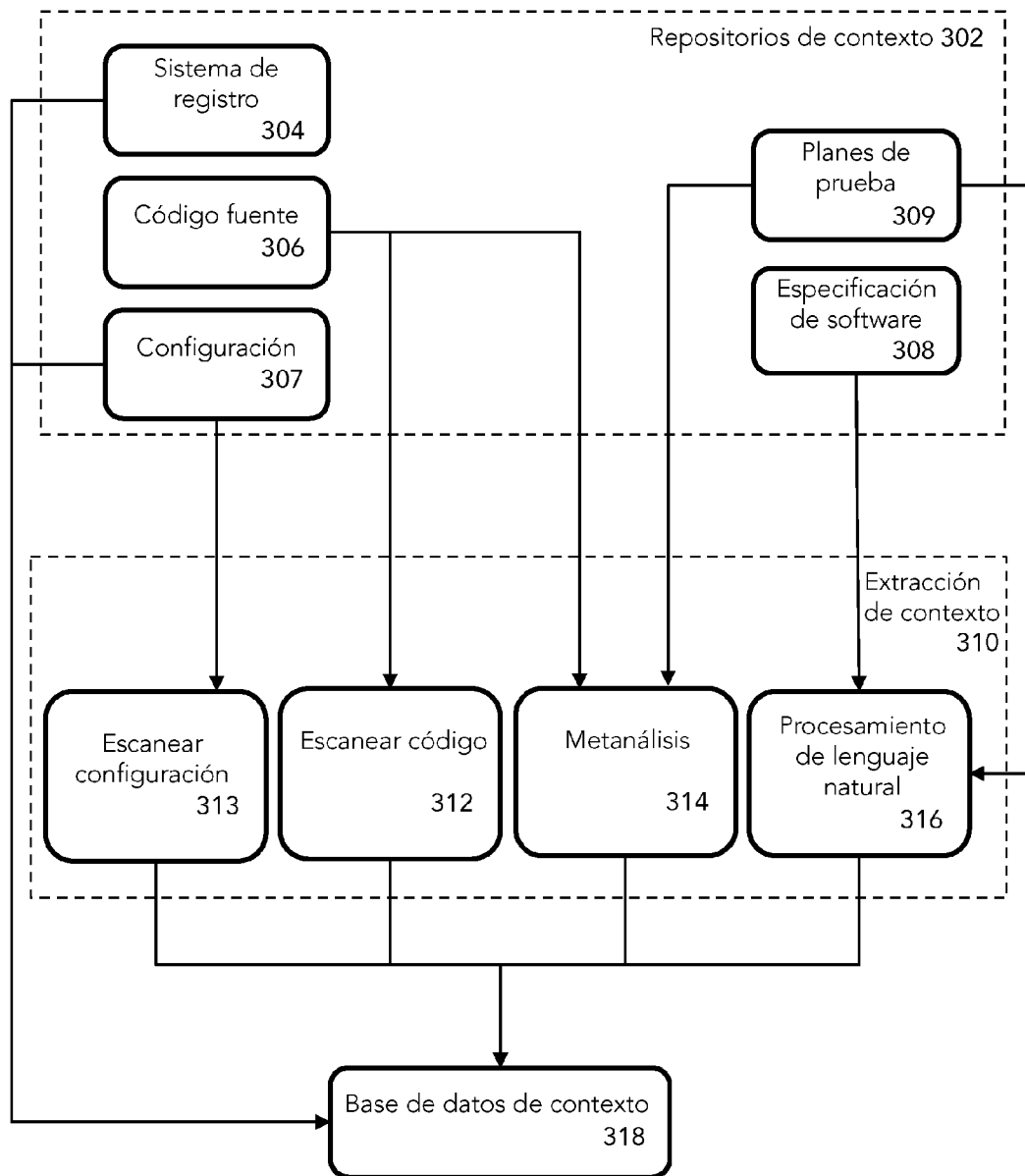


Figura 3

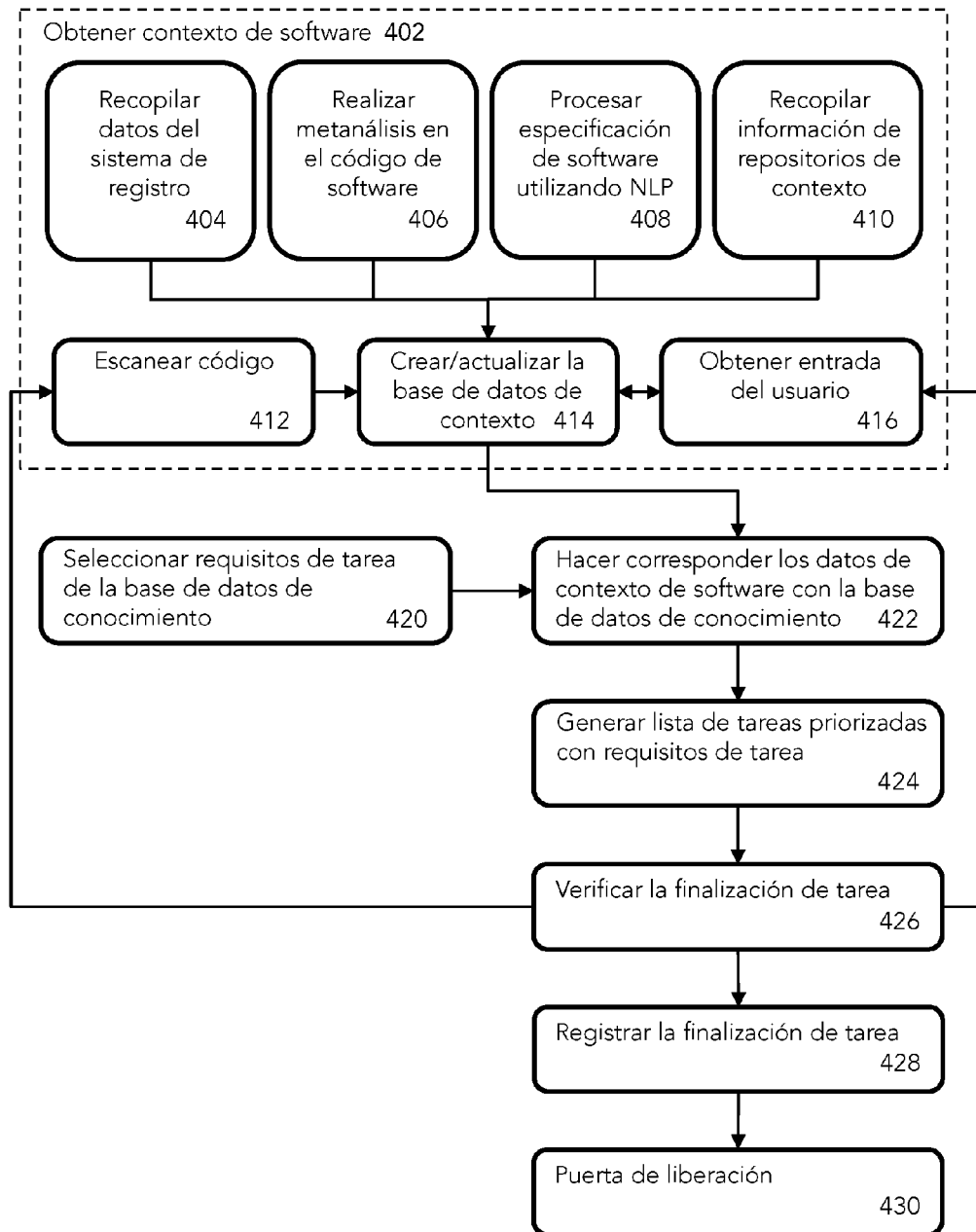


Figura 4

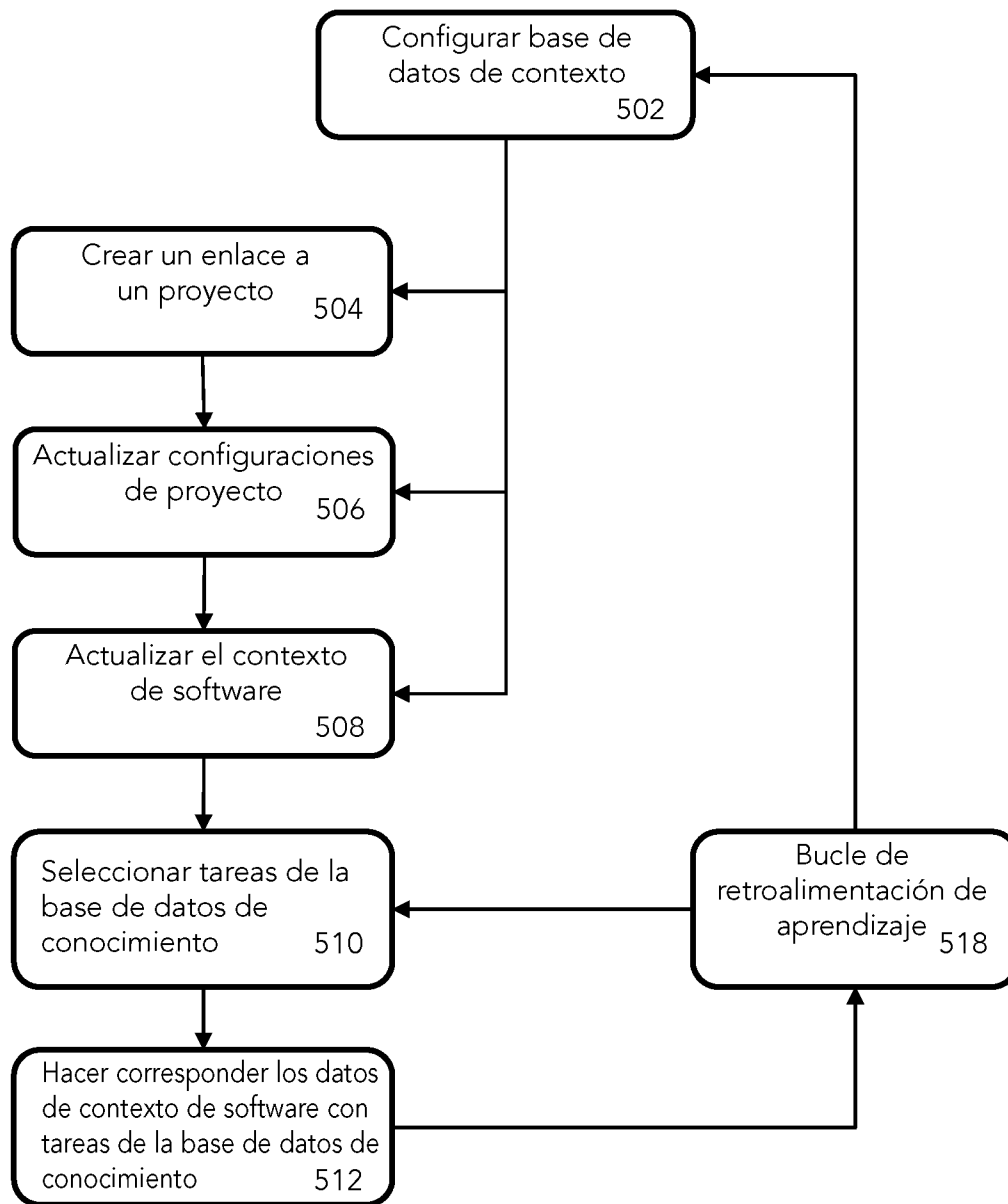


Figura 5

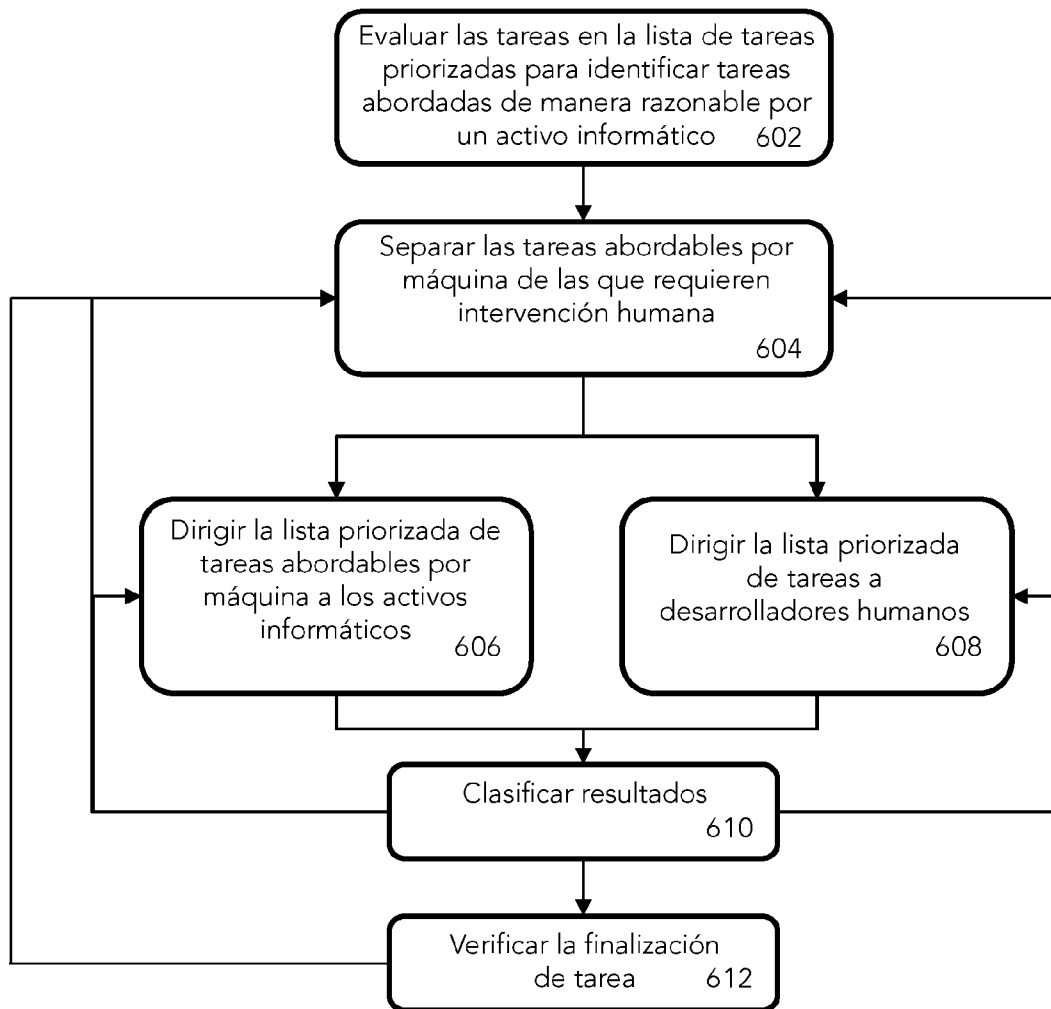


Figura 6

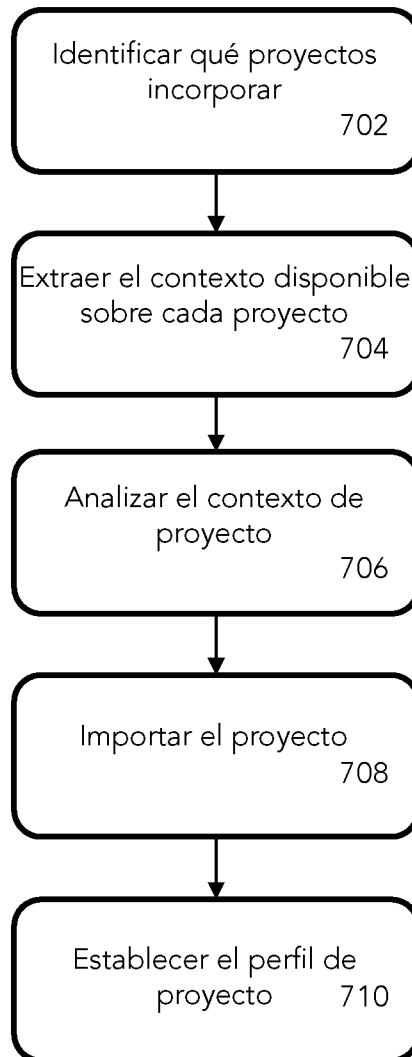


Figura 7