risti
Revista Ibérica de Sistemas e Tecnologias de Informação
Revista Ibérica de Sistemas y Tecnologías de Información

Recebido/Submission: 22/12/2022
Aceitação/Acceptance: 10/02/2023

Contrato Inteligente para la Gestión de Requerimientos en la Construcción de Software

Wilson Soto[1]

wsoto@poligran.edu.co

[1] Institución Universitaria Politécnico Grancolombiano, Bogotá, 110231, Colombia.

DOI: 10.17013/risti.49.147–160

Resumen: Uno de los procesos esenciales en la industria de software, es la negociación de los términos de contratación para la construcción de un software. El contrato de construcción de software es un contrato entre un cliente y un desarrollador, donde el desarrollador crea y entrega una pieza especifica y personalizada de software al cliente. El contrato de construcción de software es un acuerdo legal vinculante que contiene roles y responsabilidades, tiempos de entrega y pagos, criterios de aceptación, eventualidades en casos de fallas o retrasos y procedimientos para ajustes. En adición, en el contrato son especificados los requerimientos para la construcción del software. Este artículo propone el diseño y construcción de un contrato inteligente desplegado en una red blockchain para la gestión de los requerimientos dentro de una metodología ágil de construcción de software. Incluso el contrato inteligente propuesto puede ser usado para los pagos contractuales de cada requerimiento aceptado por el cliente.

Palabras-clave: Blockchain; Contratos Inteligentes; Ethereum; Procesos de Software.

Smart Contract for Requirements Management in Software Development

Abstract: One of the essential processes in the software industry is the negotiation of contract terms for the development of software. The software development contract is a contract between a customer and a developer, whereby the developer creates and delivers a specific, custom piece of software to the customer. The software development contract is a binding legal agreement that contains roles and responsibilities, delivery and payment times, acceptance criteria, eventualities in cases of failure or delay, and procedures for adjustments. In addition, the contract specifies the requirements for software development. This article proposes the design and implementation of a smart contract deployed in a blockchain network for requirement management within an agile software development methodology. Even the proposed smart contract is used for the contractual payments of each requirement accepted by the client.

Keywords: Blockchain; Ethereum; Smart Contracts; Software Processes.

1. Introducción

La tecnología blockchain es una de las tecnologías con mayor avance en los últimos años debido a sus principales características: infraestructura de computación distribuida, descentralización, inmutabilidad y una garantía de seguridad gracias a la criptografía.

En la tecnología blockchain, los mineros compiten para acumular transacciones, resolviendo el problema de dificultad computacional con el propósito de lograr consenso adicionando las transacciones con bloques a la red blockchain. La primera aplicación de la tecnología blockchain fue el Bitcoin, el cual sin un mecanismo centralizado los usuarios pueden transferir de forma segura dinero. Pero ahora varias plataformas de blockchain tales como Ethereum y Hyperledger han sido propuestas en otros campos más allá de las criptomonedas, como son los contratos inteligentes.

Los contratos inteligentes son programas descentralizados sobre la red blockchain. Las características de los contratos inteligentes son relevantes por el impacto que genera en diversas áreas. Un resumen completo de las aplicaciones basadas en contratos inteligentes se puede encontrar en (Hewa et al., 2021), donde explora entre otras: aplicaciones financieras (gestión de monedas, conocimiento del cliente, seguros, préstamos, procedimientos de auditoría y servicio de mercado de bolsa); cuidado de la salud y servicios relacionados (gestión de información de la salud, protección de datos de investigación clínica, monitoreo automático de pacientes y tratamientos); gestión de identidad y control de acceso (protección de identidad de datos, gestión de identidad descentralizada y políticas de seguridad en accesos de control); propiedad raíz (mejora de seguridad en procesos transaccionales, procesos de impuestos, comisiones por transacciones y tiempo en procesamiento); leyes y e-gobierno (ley, contractos, servicios públicos y democracia nacional); internet de las cosas (contratos inteligentes para compartir fuentes de IoT – *Internet of Things* -, vehículos independientes y ciudad inteligentes); servicios de telecomunicaciones (fuentes compartidas y autónomas en telecomunicaciones, control y gestión de acceso con contratos inteligentes, contratos inteligentes para servicios de *roaming*); gestión de logística (calidad del servicio en carga aérea y de mar, cadena de suministro en la agricultura y trazabilidad en la gestión de la cadena de suministros); contratos inteligentes en la industria (intercambio de energía, industria automotriz, protección ambiental, gestión de la construcción y gestión de tráfico aéreo).

Este artículo presenta el diseño, implementación, pruebas y despliegue de un contrato inteligente para la gestión de requerimientos en el proceso de desarrollo de software. El propósito de contrato inteligente es supervisar los requerimientos, los términos y condiciones del contrato de servicio y garantizar transparencia que no van a hacer modificados por terceros, todo como código programable, incluso las penalidades impuestas. En este sentido, los contratos inteligentes son una solución ideal para reemplazar los contratos clásicos y mejorar la eficiencia, efectividad y seguridad (Hewa et al., 2021). En definitiva, esta aproximación de contrato inteligente como herramienta de gestión de requerimientos dentro del proceso de desarrollo de software puede clasificarse como una aplicación de los contratos inteligentes en acuerdos contractuales para la construcción de software. Principalmente, en la etapa de ingeniería de requerimientos del ciclo de vida del software, donde la modificación de requerimientos es una actividad recurrente.

Esta investigación también provee una ayuda para las personas que quieran diseñar contratos inteligentes o mejorar los procesos de la industria del software.

El artículo esta estructurado de la siguiente forma: la siguiente sección muestra los antecedentes necesarios para el entendimiento de la investigación. La metodología explica el ambiente necesario, el proceso de diseño, la implementación y las pruebas del contrato inteligente como solución a un acuerdo contractual para la construcción de un software. La sección de resultados muestra el despliegue del contrato inteligente en una red blockchain. La última sección muestra las conclusiones de la investigación.

2. Antecedentes

Hoy en día la red *blockchain* es una de las tendencias tecnológicas en la industria. Esta red es una base de datos descentralizada y organizada por bloques que son inmutables. Uno de los usos más extensos de aplicaciones sobre la red blockchain son las criptomonedas, por ejemplo, *Bitcoin*. Pero otra aplicación importante es el contrato inteligente – en inglés *Smart Contract* (*SC*) -. Un contrato inteligente fue definido en (Szabo, 1994) como: "un protocolo de comunicación computarizada que ejecuta los términos de un contrato" o también se puede definir como una pieza de código que automáticamente hace respetar los términos entre las partes minimizando la participación de terceros. Muchos autores han contribuido en revisiones sistemáticas sobre las aplicaciones y trabajos a futuro sobre la tecnología blockchain. Por ejemplo, en (Casino et al., 2019) aparece un estudio exploratorio y un análisis descriptivo de la tecnología. La importancia de la investigación radica en la clasificación acorde a áreas temáticas y la taxonomía de aplicaciones propuesta.

Una de las redes más conocidas con la tecnología blockchain para desplegar contratos inteligentes es Ethereum.

2.1. Ethereum

Es una de las plataformas de *blockchain* más usadas hoy en día y reconocida por su criptomoneda nativa Ether, además de ser la primera en permitir desplegar contratos inteligentes. La red Ethereum también puede ser usada para la construcción de varias aplicaciones descentralizadas (Khan et al., 2021).

Las ventajas de la plataforma Ethereum entre otras son: sistema abierto, amplia comunidad desarrollo y disponibilidad en modo público y privado. En contraste, sus desventajas son: la sobrecarga de almacenamiento, demora en el tiempo de aprobación de las transacciones, costos altos por transacción, soporte de un solo lenguaje de programación (*solidity*) y limitaciones en la integración con otras tecnologías (Hewa et al., 2021).

2.2. Contratos inteligentes

El contrato inteligente o *smart contract* es una pieza de código que automáticamente hace respetar los términos entre las partes minimizando la participación de terceros. Un contrato inteligente desplegado en una red con tecnología *blockchain* garantiza su transparencia y seguridad.

Recientemente, varios artículos describiendo investigaciones relacionadas con contratos inteligentes se han presentado como en (Hu et al., 2021). En particular, el artículo muestra la literatura y recursos acerca de los contratos inteligentes entre 2008 y 2020. Además, resume varios desafíos e investigaciones futuras en el área. Otro artículo interesante es (Alharby et al., 2018), donde se clasifico por plataformas, áreas y aplicaciones los contratos inteligentes. Particularmente, se encontraron 40 artículos en el área de la ingeniería de software; 120 artículos relacionados con diversas aplicaciones entre las que se destacan: el internet de las cosas, computación en la nube, sector salud, sector financiero, comercio en la bolsa de valores y gestión de activos. Los autores en (Wang et al., 2018) sugirieron un estudio de contratos inteligentes, incluyendo *frameworks*, mecanismos de operación, plataformas y lenguajes de programación. Además, aplicación de escenarios, desafíos y tendencias de desarrollo futuro. Otro artículo con aportes interesantes es (Kim & Ryu, 2020) donde clasifican 391 artículos relacionados con contratos inteligentes en tres grupos: análisis dinámico, detección de vulnerabilidades y correctitud de programas en análisis estático.

Incluso, los contratos inteligentes se han aplicado como solución a un sistema de exámenes de una universidad (Kumar Samanta et al., 2021).

2.3. Aplicaciones descentralizadas

Una aplicación descentralizada – en inglés *Decentralized Application DApp* – es una aplicación que usa un contrato inteligente a través de una interfaz de presentación o *front-end*. Las aplicaciones descentralizadas se caracterizan por las siguientes cuatro propiedades: código abierto, soporte interno por criptomonedas, consenso descentralizado y puntos no centralizados de falla (Cai et al., 2018). Una metodología para construir una Dapp consiste en combinar el diseño e implementación de un contrato inteligente, el despliegue en una red blockchain y la construcción de una aplicación web para interactuar.

2.4. Contrato de desarrollo de software

En la industria del software, un contrato de desarrollo de software (a veces conocido como acuerdo o contrato de servicios maestro) es un contrato entre dos personas: un cliente y un proveedor de software. El cliente determina las características y funcionalidades requeridas y el proveedor construye el software a la medida a cambio de un pago o precio acordado.

Generalmente, un contrato de desarrollo de software incluye, pero no se limita a las siguientes secciones:

1. Compromisos del desarrollador
2. Responsabilidades del cliente
3. Funcionalidades del software
4. Remuneraciones y pagos
5. Términos y vencimiento del contrato
6. Cláusulas de confidencialidad
7. Derechos de propiedad intelectual

8. Garantías y descargos de responsabilidades
9. Cláusula de no competencia
10. Cláusula de funcionalidad
11. Condiciones generales

2.5. Ingeniería de requerimientos

En el proceso de desarrollo de software, los requerimientos son una etapa clave y frecuentemente la primera. Debido a la rigurosidad y al proceso sistemático que se debe realizar en esta etapa, se conoce como la etapa de la ingeniería de requerimientos. Usualmente, la ingeniería de requerimientos envuelve tres etapas: elicitación, especificación y validación. En especial, uno de los objetivos en la etapa de especificación es escribir los requerimientos en un documento. Este documento de requerimientos es un documento esencial en la construcción de software, incluso para pagar a las personas responsables que lo desarrollan (Sommerville, 2016).

3. Metodología

El contrato inteligente presentado en este artículo puede ser aplicado en una metodología ágil como en (Udvaros et al., 2023), donde el contrato inteligente en la metodología ágil scrum actúa como un artefacto de pila de producto – *producto backlog* - construido por el propietario de producto – *producto owner* –. Además de contener los requerimientos (historias de usuario) del software, las modificaciones realizadas a los requerimientos se registran en el contrato inteligente funcionando como una herramienta de gestión que garantiza la propiedad de la transparencia por estar desplegado en la red blockchain. Incluso el cliente puede validar las historias de usuario terminadas después de cada *sprint* y hacer los pagos contractuales correspondientes (Figura 1).

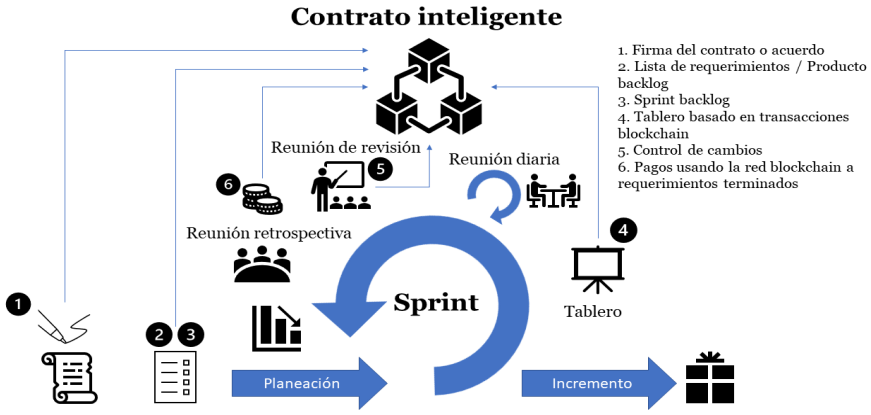


Figura 1 – Aplicación de contrato inteligente como herramienta de gestión de requerimientos en un marco de trabajo Scrum

A continuación, se describe cada una de las etapas del proceso de construcción del contrato inteligente.

3.1. Diseño

En esta fase fueron identificados los actores y las funciones. Los actores son el cliente y el desarrollador. Las funciones identificadas son: la firma del contrato, términos de terminación del contrato y la gestión de los requerimientos (cambios, pagos, tiempos de entrega, ajustes y criterios de aceptación) entre otros (Tabla 1).

Función	Parámetros	Retorno	Descripción
signature	-	-	Firma del acuerdo
initTime	-	uint	Inicio del acuerdo
nowDate	-	uint	Fecha actual
endTime	-	uint	Finalización del acuerdo
showTotalTime	-	uint	Tiempo del contrato
addRequirement	_reqID, _desc	-	Adición de requerimiento
updateRequirement	_reqID, _desc	(string, uint, , ,)	Modificación de requerimiento
getRequirement	_reqID	(string, uint, string, bool,unit)	Consulta de requerimiento
validationRequirement	_reqID	(string, uint, string, bool, ,)	Validación de requerimiento
acceptanceCriteria	_reqID, _desc	-	Criterios de aceptación por requerimiento
validateMinimumPay	valueUsd	-	Valor mínimo en dólares (USD) por transacción
getVersion	-	uint256	Versión de la dirección con la conversión ETH/USD
getPrice	-	uint256	Valor actual de 1 Ether (ETH) en dólares (USD)
getConversionRate	ethAmount	uint256	Conversión de USD a ETH.
fundAccount	-	-	Deposito en cuenta
withdrawAccount	-	-	Retiro en cuenta

Tabla 1 – Funciones del contrato inteligente propuesto

3.2. Implementación

Dos implementaciones fueron necesarias para construir el contrato inteligente. El código está disponible en el siguiente repositorio público: https://github.com/Volfra/SmartContracts/tree/main/test

La primera implementación, el contrato inteligente (SWAgreement.sol) usando el lenguaje solidity sobre el IDE Remix y Visual Studio Code y la segunda implementación, una aplicación descentralizada DApp usando el lenguaje Python (deploy.py - fund_and_withdraw.py) sobre Visual Studio Code.

3.3. Pruebas

Se requiere la instalación y configuración de Ganache para pruebas locales y la configuración de una billetera virtual para el despliegue en la red Ethereum. La billetera virtual fue creada en *Metamask* sobre la red de prueba Goerli (Figura 2).

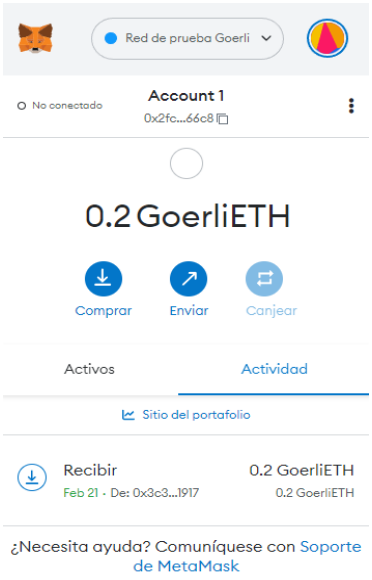


Figura 2 – Billetera virtual en *Metamask*

3.4. Despliegue

El contrato inteligente es desplegado en la red Ethereum. La red Ethereum proporciona una red de prueba llamada *Goerli*. Para ello es necesario abrir una cuenta sobre *Infura*. También es necesario *Brownie* para la aplicación descentralizada DApp. Otra herramienta usada es *Etherscan*, que explora la red Ethereum para validar las transacciones de los contratos inteligentes (https://goerli.etherscan.io/).

4. Resultados

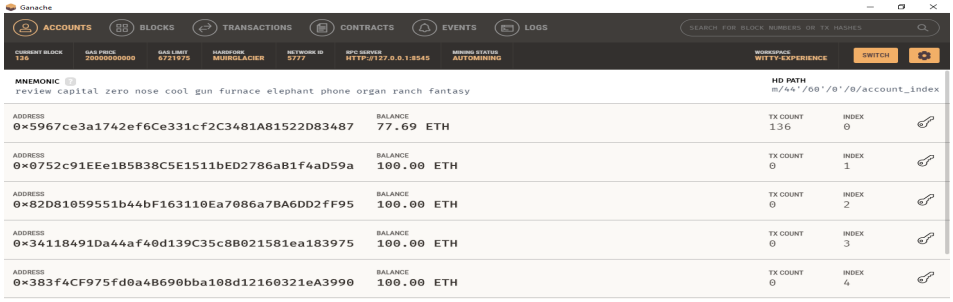
El contrato inteligente fue probado sobre un ejemplo de requerimiento basado en el *framework* Scrum – historia de usuario y criterios de aceptación –. Cada uno de los siguientes pasos genera una transacción y bloque en la red:

1. Despliegue y firma del contrato entre el desarrollador y el cliente.
2. Adición de la historia de usuario UH1.
3. Como cliente necesito hacer una compra así puedo tener comida en mi casa
4. Modificación de la historia de usuario UH1.
    - Como cliente necesito hacer una orden así puedo tener comida en mi casa

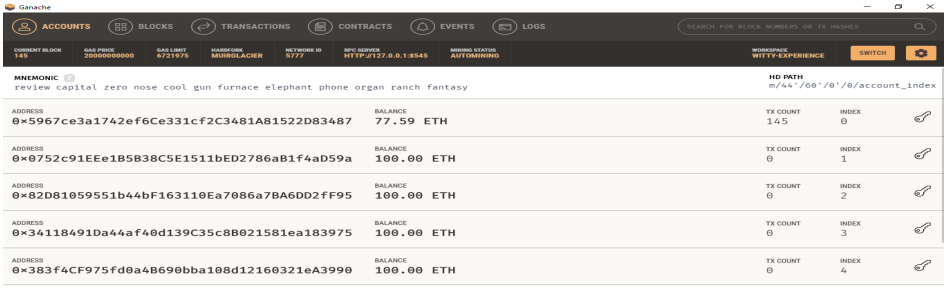
5. Adición de criterios de aceptación a la historia de usuario.
   - ¿Puedo ver el total acumulado de lo elegido hasta el momento?
   - ¿Puedo cambiar mi orden antes que pague?
   - ¿Puedo hacer una orden a cualquier hora (7x24)?
6. Consulta de la historia de usuario UH1.
7. Validación de la historia de usuario UH1.

Pago de la historia de usuario UH1 o requerimiento. El pago está alrededor de 164.3 USD. For the date, 1 ETH = $1643,00 USD, therefore, the payment is around 0.1 ETH or 100000000000000000 Wei

La primera prueba es sobre un ambiente local usando Ganache. La dirección con índice 0 es una cuenta con un monto de 77.69 ETH (figura 3(a)), que posterior a la ejecución de la prueba anterior, es decir, creado, modificado, validado y pagado el requerimiento su monto se reduce en 0.1 ETH, simulando un proceso completo (figura 3(b)). Además, se pueden observar todos los bloques (figura 4) y las transacciones realizadas por la prueba. En la figura se muestran los detalles de las transacciones para el despliegue del contrato y pago de requerimiento (figura 5). En la transacción del pago se puede observar el valor consignado a la dirección donde se desplegó el contrato.



(a)



(b)

Figura 3 – Billeteras de prueba con Ganache. La billetera usada es la de índice 0 desde donde se hace el pago al contrato. (a) antes del pago. (b) después del pago.

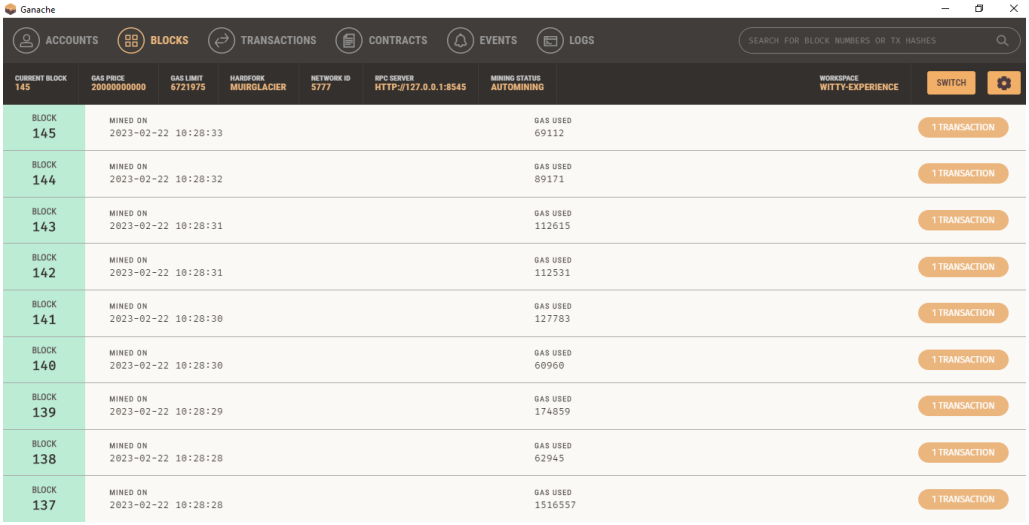


Figura 4 – Bloques generados después de realizar la prueba sobre el ambiente de prueba local usando Ganache.

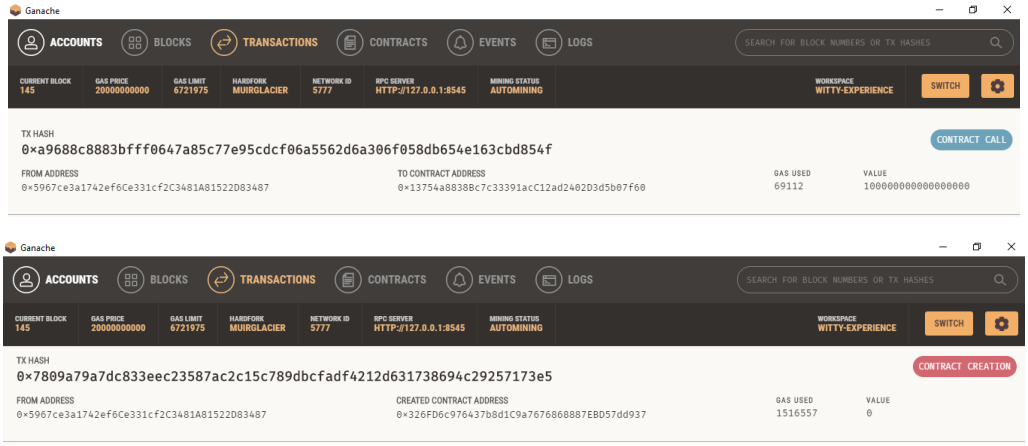


Figura 5 – Detalle de las transacciones en el despliegue del contrato y pago del requerimiento.

La segunda prueba es la ejecución del contrato inteligente sobre Infura y la red de pruebas Goerli. Las transacciones vistas desde la aplicación descentralizada (figura 6). La dirección del contrato desplegado en la red es: 0x81cDAD9BC22Da4dA3b6E62090e9a248Eb7753E9b

```
Brownie v1.19.3 - Python development framework for Ethereum

TestProject is the active project.

Running 'scripts\deploy.py::main'...

Wallet (Wei):  205095104534387702

Transaction sent:
0xaf0e0f60bb36c6a2618150c6bb3c308d3c354079766edc0a29b34d5cc6455bdf

  Gas price: 30.811919089 gwei   Gas limit: 1681852   Nonce: 11

  SWAgreement.constructor confirmed   Block: 8540260   Gas used: 1528957
(90.91%)

  SWAgreement deployed at: 0x81cDAD9BC22Da4dA3b6E62090e9a248Eb7753E9b

Address contract: 0x81cDAD9BC22Da4dA3b6E62090e9a248Eb7753E9b

Address wallet deploy contract client
0x2fcd44991ca6B22177Dc6b45e8699C6C389066c8

Address wallet client 0x2fcd44991ca6B22177Dc6b45e8699C6C389066c8

Transaction sent:
0xcd8b137fea385e85bf862290c9c87dc92e4c9f1df6d78312de277a0255a82984

  Gas price: 29.207733731 gwei   Gas limit: 74519   Nonce: 12

  SWAgreement.signature confirmed   Block: 8540268   Gas used: 70545
(94.67%)

Agreement signature Date (2023, 2, 23)

Transaction sent:
0xabaa6bf92fe985c02f115d1ebfebc6fd62b145770753872e0e6ba8d40c9fe2ab

  Gas price: 27.550121552 gwei   Gas limit: 206644   Nonce: 13

  SWAgreement.addRequirement confirmed   Block: 8540282   Gas used: 187859
(90.91%)

Create User History ('UH1', 1677128100, 'Como cliente necesito hacer una
compra así puedo tener comida en mi casa', False, 1677128100, ())

Transaction sent:
0x27696f95a471922de6510d856f3dbb98911ecd295e17b0f75e7f6a55e6e57741

  Gas price: 26.739426503 gwei   Gas limit: 68461   Nonce: 14

  SWAgreement.updateRequirement confirmed   Block: 8540283   Gas used: 60060
(87.73%)

Update User History ('UH1', 1677128100, 'Como cliente necesito hacer una
orden así puedo tener comida en mi casa', False, 1677128100, ())

Transaction sent:
0xee474f920d19a1ff936273700f4c6b9be4b8dffec3f4f1c63415022affe21567

  Gas price: 25.538481819 gwei   Gas limit: 149581   Nonce: 15

  SWAgreement.acceptanceCriteria confirmed   Block: 8540284   Gas used:
135983 (90.91%)
```

```
Transaction sent:
0x779fe54e33074d3dfdc2e4020292d61baddb2253e3d572cd1f912071c722b371

  Gas price: 25.254563274 gwei   Gas limit: 130494   Nonce: 16

  SWAgreement.acceptanceCriteria confirmed   Block: 8540285   Gas used:
118631 (90.91%)

Transaction sent:
0xf1bbc43f64e2d72cf580cefc0dd396186cf72ef62fb1054d4c3cb5ae8f736564

  Gas price: 25.254563274 gwei   Gas limit: 130586   Nonce: 17

  SWAgreement.acceptanceCriteria confirmed   Block: 8540286   Gas used:
118715 (90.91%)

('UH1', 1677128100, 'Como cliente necesito hacer una orden así puedo tener
comida en mi casa', False, 1677128100, ((1, '¿Puedo ver el total acumulado
de lo elegido hasta el momento?'), (2, '¿Puedo cambiar mi orden antes que
pague?'), (3, '¿Puedo hacer una orden a cualquier hora (7x24)?')))

Transaction sent:
0x3d892b3a5d1937f9908ba7639e08aa3406fac14b51b4f1a6d5e2381eff5b2caa

  Gas price: 23.188855851 gwei   Gas limit: 115798   Nonce: 18

  SWAgreement.validationRequirement confirmed   Block: 8540290   Gas used:
105271 (90.91%)

('UH1', 1677128100, 'Como cliente necesito hacer una orden así puedo tener
comida en mi casa', True, 1677128220, ((1, '¿Puedo ver el total acumulado
de lo elegido hasta el momento?'), (2, '¿Puedo cambiar mi orden antes que
pague?'), (3, '¿Puedo hacer una orden a cualquier hora (7x24)?')))

Pago requerimiento UH1 x 164.3 USD (21.02.23) = 0.1 ETH

Transaction sent:
0xd149c74f466a96449dccd56c9f605863c4c7d8c2c034d88b22a09c5f2c39bd48

  Gas price: 22.904947585 gwei   Gas limit: 79543   Nonce: 19

  SWAgreement.fundAccount confirmed   Block: 8540295   Gas used: 75112
(94.43%)
```

Figura 6 – Transacciones generadas después de realizar la prueba sobre la red Goerli en Infura desde la aplicación descentralizada DApp.

Etherscan es usado para observar las transacciones del contrato en la red Ethereum (figura 7). Es posible identificar en las transacciones la creación del contrato, la firma y el pago del requerimiento. Todas las transacciones tienen una dirección – billetera del cliente -, el valor y la tarifa de la transacción. La tarifa de transacción, tarifa de gas o precio de gas que significa el valor a pagar por los recursos computacionales sobre la red Ethereum y es calculado al multiplicar el precio de gas por el gas usado por la transacción.

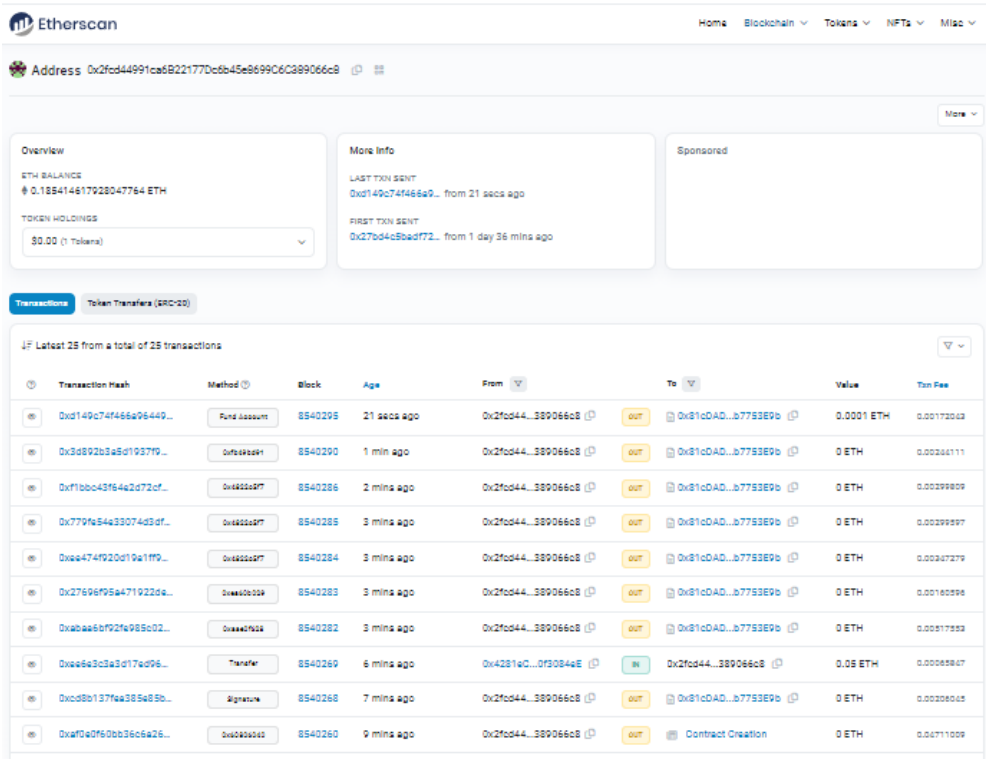


Figura 7 – Transacciones generadas después de realizar la prueba sobre la red Goerli en Infura. Las transacciones pueden ser vistas en el enlace https://goerli.etherscan.io/address/0x2fcd44991ca6b22177dc6b45e8699c6c389066c8

En el caso en particular, el valor total de todas las tarifas de transacciones es alrededor de 0,070238890 ETH o 7 USD.

También, Etherscan permite la inspección de los detalles de cada transacción, por ejemplo, el detalle de la transacción del pago del requerimiento (figura 8).

5. Conclusiones

Este artículo mostró un contrato de construcción de software como contrato inteligente sobre la red blockchain. Se incluyó el diseño, implementación, pruebas y despliegue del contrato inteligente sobre la red Ethereum. Una solución de este tipo puede llegar a ser una herramienta importante en la etapa de ingeniería de requerimientos dentro del ciclo de vida del software. Principalmente conociendo los desafíos en la ingeniería de requerimientos como su constante evolución y trazabilidad durante todo el ciclo, por lo tanto, herramientas de gestión que contribuyan con el mejoramiento de los procesos en la construcción de software son valiosas.

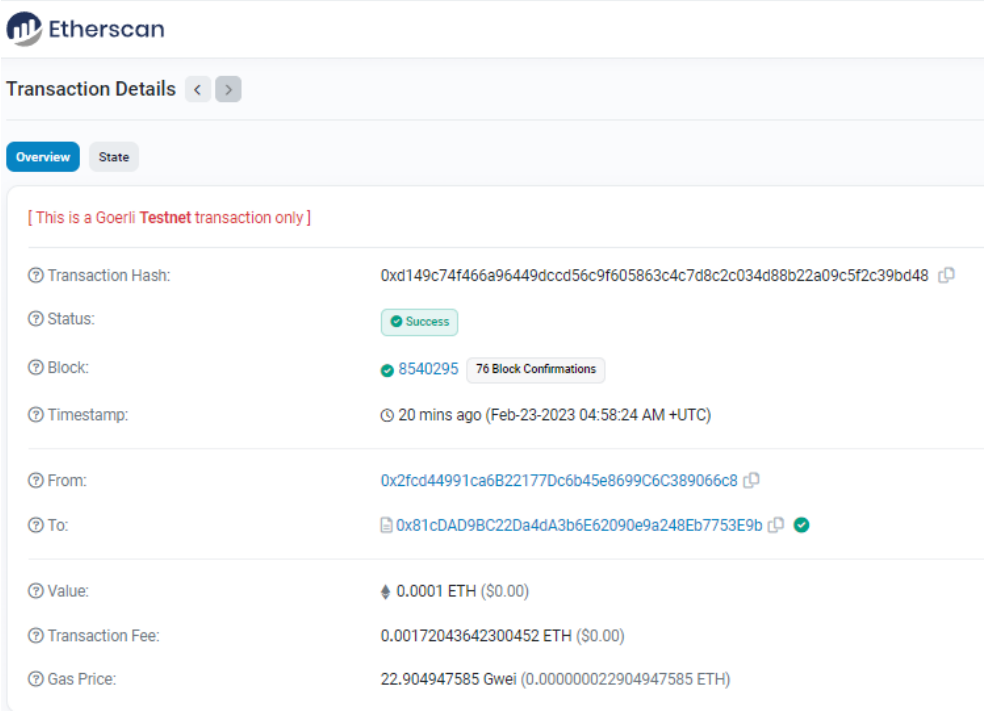


Figura 8 – Detalle de la última transacción después de realizar la prueba sobre la red Goerli en Infura. El detalle de la transacción puede ser visto en el enlace https://goerli.etherscan.io/tx/0xd149c74f466a96449dccd56c9f605863c4c7d8c2c034d88b22a09c5f2c39bd48

Los contratos inteligentes desplegados en una red blockchain y usados dentro de la etapa de ingeniería de requerimientos en el ciclo de vida de software tienen entre otras ventajas: precisión, transparencia, seguridad, confianza y no intermediación en la gestión de los requerimientos (tiempos de entrega y pagos, ajustes, riesgos, control de cambios y criterios de aceptación).

El trabajo futuro puede focalizarse en la implementación de contratos inteligentes en diversas etapas del ciclo de vida del software, como también evaluar el impacto de este tipo de herramientas en la calidad del software.

Recomendable investigar el uso de otro tipo de redes con tecnología blockchain para desplegar contratos inteligentes, como por ejemplo aquellas con transacciones sin tarifa.

Referencias

Alharby, M., Aldweesh, A., & van Moorsel, A. (2018). Blockchain-based Smart Contracts: A Systematic Mapping Study of Academic Research (2018). International Conference on Cloud Computing, Big Data and Blockchain, ICCBB 2018. https://doi.org/10.1109/ICCBB.2018.8756390

Cai, W., Wang, Z., Ernst, J. B., Hong, Z., Feng, C., & Leung, V. C. M. (2018). Decentralized Applications: The Blockchain-Empowered Software System. *IEEE Access*, *6*, 53019–53033. https://doi.org/10.1109/ACCESS.2018.2870644

Casino, F., Dasaklis, T. K., & Patsakis, C. (2019). A systematic literature review of blockchain-based applications: Current status, classification and open issues. *Telematics and Informatics*, *36*, 55–81. https://doi.org/10.1016/J.TELE.2018.11.006

Hewa, T., Ylianttila, M., & Liyanage, M. (2021). Survey on blockchain based smart contracts: Applications, opportunities and challenges. *Journal of Network and Computer Applications*, *177*, 102857. https://doi.org/10.1016/J.JNCA.2020.102857

Hu, B., Zhang, Z., Liu, J., Liu, Y., Yin, J., Lu, R., & Lin, X. (2021). A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. *Patterns*, *2*(2), 100179. https://doi.org/10.1016/J.PATTER.2020.100179

Karamitsos, I., Papadaki, M., Barghuthi, N. B. al, Karamitsos, I., Papadaki, M., & Barghuthi, N. B. al. (2018). Design of the blockchain Smart Contract: A Use Case for Real Estate. *Journal of Information Security*, *9*(3), 177–190. https://doi.org/10.4236/JIS.2018.93013

Khan, S. N., Loukil, F., Ghedira-Guegan, C., Benkhelifa, E., & Bani-Hani, A. (2021). blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-Peer Networking and Applications*, *14*(5), 2901–2925. https://doi.org/10.1007/S12083-021-01127-0

Kim, S., & Ryu, S. (2020). Analysis of blockchain Smart Contracts: Techniques and Insights. *Proceedings - 2020 IEEE Secure Development, SecDev 2020*, 65–73. https://doi.org/10.1109/SECDEV45635.2020.00026

Kumar Samanta, A., Bidyut, &, Sarkar, B., & Chaki, N. (2021). A blockchain-Based Smart Contract Towards Developing Secured University Examination System. *Journal of Data, Information and Management 2021 3:4*, *3*(4), 237–249. https://doi.org/10.1007/S42488-021-00056-0

Sommerville, I. (2016). Software Engineering Tenth Edition. *Software Engineering Tenth Edition*, 111–133.

Szabo, N., 1994. Smart contracts.

Udvaros, J., Forman, N., & Avornicului, S. M. (2023). Agile Storyboard and Software Development Leveraging Smart Contract Technology in Order to Increase Stakeholder Confidence. *Electronics 2023, Vol. 12, Page 426*, *12*(2), 426. https://doi.org/10.3390/ELECTRONICS12020426

Wang, S., Yuan, Y., Wang, X., Li, J., Qin, R., & Wang, F. Y. (2018). An Overview of Smart Contract: Architecture, Applications, and Future Trends. *IEEE Intelligent Vehicles Symposium, Proceedings*, *2018-June*, 108–113. https://doi.org/10.1109/IVS.2018.8500488