Complex Systems Oriented Approach for dApps Analysis

Giuseppe Destefanis
Department of Computer Science
Brunel University London, UK
giuseppe.destefanis@brunel.ac.uk

***Abstract*—This position paper discusses on applying complex systems theory to analyse decentralised applications (dApps). It highlights the need for understanding smart contract networks' dynamics that are at the base of such platforms. Through a discussion that includes architecture, use cases, and both network and function-level scrutiny, including traditional and new software metrics, we argue how insights from complex systems can improve the understanding of dApps' behavior, vulnerabilities, and areas for optimisation. This position paper lays a foundation for enhancing the resilience, efficiency, and security of dApps within the blockchain environment.**

***Index Terms*—complex systems, smart contracts, dApps, blockchain**

I. Introduction

Complex systems are a cross-disciplinary area of science focusing on the study of systems where numerous components interact in different ways, producing behaviours that are not easily predicted by examining the components alone. These systems are distinguished by their vast set of different, interconnected elements and complex organisational frameworks.

In the context of computer science and information technology, complex systems can refer to computational networks where nodes represent distinct computational entities and edges denote the interactions or data flows between them. This approach is particularly relevant when analysing decentralised applications (dApps) on blockchain networks like Ethereum, where smart contracts act as the building blocks of a "complex" system. By modeling the relationships and interactions between smart contracts as a complex network, researchers can better understand the functionality, dependencies, and potential vulnerabilities within the dApp ecosystem.

Parisi [9] defines Complex systems in the following way: "*Generally speaking, we could say that a complex system can stay in many different equilibrium states, while a simple system may stay only in one or a few equilibrium states. For example, an animal like a dog can do many different actions (e.g., play, sleep, eat, hunt); an animal can switch from one state to another state in a very short time ad the effect of a small perturbation, e.g., suddenly waking up after hearing a suspicious noise. In a nutshell what does not change over time (or change irreversibly) is not complex, while a system that can take many different forms or behaviors certainly is.*."

A **complex system** is a system composed of interconnected parts that as a whole exhibit one or more properties (behavior among the possible properties) not obvious from the properties of the individual parts. This concept is interesting for the study of decentralised systems and blockchain technology, as it helps in capturing the non-linear interactions and the collective behavior of smart contracts within dApps, which emerge from the interplay of their constituent smart contracts.

This position paper aims to examine how applying complex systems theory can improve our understanding of decentralised applications. Through the lens of a case study dApp similar to Uniswap[1], an extensive, open marketplace on the Ethereum blockchain for cryptocurrency exchanges without traditional goods or services, we will explore how this theoretical approach can yield more profound insights into dApps' architecture and behaviours. It stands as a paradigm of decentralisation, operating autonomously without any central authority, and is instead driven by a network of smart contracts and active user engagement.

To grasp the workings of this Uniswap-like dApp, let us consider the following components.

Automated Market Making (AMM): unlike traditional markets that rely on buyers, sellers, and sometimes intermediaries to facilitate matches, this platform employs AMM to eliminate the need for a middleman. It uses liquidity pools for each cryptocurrency pair (e.g., Ethereum and a US Dollar Token equivalent), enabling continuous trading by using these pools of funds.

Liquidity Pools: these are essentially large reserves containing two types of cryptocurrencies. Users can contribute their assets to these pools to become liquidity providers, earning transaction fees as a reward, akin to earning interest in a savings account but with the added element of risk and higher potential returns.

Trading and Pricing: users looking to exchange one cryptocurrency for another interact directly with the liquidity pool containing both currencies. Prices are determined automatically by smart contracts based on the relative quantities of each cryptocurrency in the pool, with the potential for price shifts as trades alter the pool's balance.

Decentralisation: operating on the Ethereum blockchain and governed by smart contracts, this platform eschews a centralized authority, granting users more control. Participants can

[1]https://app.uniswap.org/swap

directly engage with the marketplace via their cryptocurrency wallets.

This conceptual dApp, inspired by Uniswap, is an exemplar of a complex system, as evidenced by its ability to inhabit multiple equilibrium states—each liquidity pool can maintain varying levels of liquidity, depth, and pricing that are dynamically adjusted based on market conditions and participant actions. Specifically, the following points illustrate its alignment with the definition of a complex system:

Interconnectedness: the dApp's numerous liquidity pools represent a network of interconnected parts, where each pool is linked to others through the shared practice of AMM and the overarching smart contract architecture. This network forms a web of financial interactions and dependencies.

Diversity of Components: each liquidity pool operates under a uniform set of rules but is distinguished by its unique assortment of assets, trading volume, and user-provided liquidity. This variety enhances the overall strength and adaptability of the system.

Adaptability: the system exhibits a high degree of adaptability, adjusting to changes such as significant trades or shifts in market sentiment. Liquidity providers can freely enter and exit pools, while traders can execute trades at any time, all of which contribute to the system's fluidity and its ability to reach new equilibrium states.

Emergent Behavior: the pricing mechanism within each pool—determined by the relative quantities of the two assets and the AMM algorithm—is a prime example of emergent behavior. It is not dictated by any single participant but rather arises from the collective action of all traders and liquidity providers.

Non-linearity: The relationship between trading activities, liquidity provision, and price is highly non-linear. Large trades can lead to significant price impacts, which in turn can trigger cascading effects as other participants respond to the new prices.

Dynamic Nature: as described by Parisi [9], the system can rapidly transition between states in response to small perturbations—a large trade or a sudden shift in market conditions can quickly alter a pool's state, much like an animal reacting to its environment.

Complex Feedback Loops: the governance model introduces complex feedback loops into the system, where token holders can propose and vote on changes that may alter the dApp's rules and parameters, influencing the behavior of all other components.

II. Decentralised Application Architecture

The software architecture of the Uniswap-like dApp is built on the Ethereum blockchain, using smart contracts to automate the functions of its decentralised exchange. This architecture uses a collection of smart contracts to enable trading, liquidity provision, and governance mechanisms. Here is a generalised overview of some key smart contracts and components that facilitate the operation of such a dApp:

Factory Contract: responsible for creating new smart contracts for each cryptocurrency pair, acting as a registry and deploying an Exchange Contract (or Pool Contract) for new market pairs as needed.

Exchange Contract (or Pool Contract): dedicated to handling trades and liquidity for specific token pairs, these contracts use the Automated Market Making (AMM) algorithm to set prices based on the current supply of the tokens within the pool.

Router Contract: serves to bridge users with Exchange Contracts, streamlining the trade execution and liquidity management by directing interactions to the correct Exchange Contracts for the user's chosen token pairs.

Liquidity Pool Tokens (LP Tokens) Contract: issues LP tokens to users who provide liquidity, representing their share in the pool. These tokens can be exchanged back for the underlying assets at any time, effectively managing the issuance and redemption process.

Governance Contract: facilitates a governance system where token holders have the power to vote on protocol changes. This contract oversees the proposal, voting, and implementation phases of governance decisions.

WETH Contract: addresses the need for trading Ethereum (ETH) in a tokenized form within pools that primarily handle ERC-20 tokens by wrapping ETH into Wrapped Ether (WETH), allowing for direct trades against other ERC-20 tokens.

Staking Contract: offers users the opportunity to stake their LP tokens or other supported tokens to earn rewards, which can be derived from trading fees or other incentive mechanisms designed within the dApp.

TimeLock Contract: introduces a delay between the approval of governance proposals and their execution, providing a window for community response to new changes.

Migration Contract: supports the dApp through version upgrades by facilitating the transfer of liquidity from one version to the next, ensuring a smooth transition for liquidity providers.

Position Manager Contract: a feature akin to what was introduced in Uniswap V3, allowing for concentrated liquidity positions within specific price ranges, managing the allocation, modification, and removal of capital.

This Uniswap-like dApp's architecture is designed to be modular, with each contract fulfilling distinct roles within the ecosystem.

III. Use Case

Let us now consider a use case within a Uniswap-like decentralised application (dApp), where a user aims to swap Ethereum (ETH) for DAI (a stablecoin pegged to the US dollar) and subsequently provide liquidity to the ETH-DAI pool. This scenario illustrates the interaction among various smart contracts to enable these actions.

- **Step 1: Swapping ETH for DAI**
  - **User interaction with Router Contract**: the user initiates the process through the dApp's interface,

specifying the amount of ETH they wish to exchange for DAI. This action interfaces with the Router Contract.
  - **WETH Contract Conversion**: to facilitate the swap within the dApp's pools, which primarily handle ERC-20 tokens, the Router Contract converts the ETH to Wrapped Ether (WETH) through the WETH Contract.
  - **Router to Exchange Contract**: the Router Contract locates the correct Exchange Contract (or Pool Contract) for the WETH-DAI pair, calculating the DAI amount the user receives based on the pool's existing WETH to DAI ratio and the AMM algorithm.
  - **Execution of Trade**: the Router Contract conducts the trade, transferring the specified WETH amount into the pool and allocating the calculated DAI amount to the user from the pool.
- **Step 2: Providing Liquidity to the ETH-DAI Pool**
  - **Token Approval**: the user decides to supply liquidity to the ETH-DAI pool, necessitating authorization for the Router Contract to manage their WETH and DAI tokens. This approval process involves interaction with the ERC-20 contracts of WETH and DAI.
  - **Adding Liquidity via Router Contract**: after specifying the desired amounts of WETH and DAI to contribute, the Router Contract verifies the existence of an Exchange Contract for the WETH-DAI pair via the Factory Contract. Upon confirmation, it aids in adding the user's tokens to the pool.
  - **LP Tokens Issued**: the Exchange Contract grants LP tokens to the user proportional to the provided liquidity, managed by the Liquidity Pool Tokens (LP Tokens) Contract, symbolising the user's share in the pool.
- **Step 3: Staking LP Tokens (Optional)**
  - **Staking Contract Interaction**: for users seeking additional rewards, staking their LP tokens through the Staking Contract is an option. This contract oversees reward distribution, potentially sourced from trading fees or specific incentives within the dApp.
  - **Earning Rewards**: users accumulate rewards over time based on their liquidity contribution, represented by the staked LP tokens.

Throughout this process:

- The **Governance Contract** may play a role if recent protocol adjustments influence fees, rewards, or other functionalities used by the user.
- The **TimeLock Contract** introduces a delay for governance vote-approved changes, offering the community a period for review.
- The **Migration Contract** becomes relevant for users transitioning their liquidity from an older dApp version to an updated one.

In the functioning of a Uniswap-like dApp, interactions with certain contracts external to the dApp itself are fundamental to its operation. A key example of such an external contract is the Wrapped Ether (WETH) contract, which adheres to the ERC-20 standard and enables the trading of Ethereum (ETH) as though it were an ERC-20 token. This functionality is vital since the liquidity pools within the dApp are configured to support ERC-20 tokens exclusively. As a result, when users wish to exchange ETH for another ERC-20 token, like DAI, they must first convert their ETH to WETH through the WETH contract. Additionally, users are required to provide explicit permission for the dApp's Router Contract to manage their WETH and DAI tokens. This permission is granted via direct interaction with the ERC-20 contracts for WETH and DAI, using the approve function to authorize the Router Contract to execute token transfers on behalf of the users. This authorization step is indispensable for empowering the Router Contract to facilitate token handling during trading and liquidity provision activities, without taking direct control of the users' assets. These interactions with external contracts highlight the significance of interoperability and composability within the Ethereum blockchain ecosystem, enabling various contracts to collaborate effectively to provide the sophisticated services characteristic of the dApp.

IV. Complex Networks

In analysing the architecture of our example, we can conceptualise its smart contracts as nodes within a complex network graph, where the edges represent the interactions between these contracts. Each node, or smart contract, serves a specific function within the ecosystem—such as facilitating exchanges (Exchange/Pool Contracts), managing liquidity (Liquidity Pool Tokens Contract), or governing the dApp's parameters (Governance Contract). The connections or edges between these nodes illustrate the flow of data, tokens, and control commands that enable the dApp's operations.

A connection from the Router Contract node to the Exchange/Pool Contract shows a trade happening, while a link to the WETH Contract is for turning Ethereum into Wrapped Ether for ERC-20 transactions. This way of mapping helps in understanding the structure and highlighting weaknesses, like potential security risks or places that could be made more efficient, such as improving trades or lowering costs.

Figure 1 shows a network of smart contract interactions within our Uniswap-like dApp, showing the interconnected nature of the system, as well as the inclusion of external contracts which introduce external dependencies and potential vulnerabilities.

In the graph, each node represents a smart contract, while the edges illustrate the interactions between them:

- The **Router Contract** is central, interfacing with the **WETH Contract** for wrapping and unwrapping Ether, and with the **Exchange/Pool Contract** for trade execution and liquidity management.
- The **Factory Contract** is tasked with creating new Exchange/Pool Contracts for different token pairs, essential for the dApp's trading functionality.

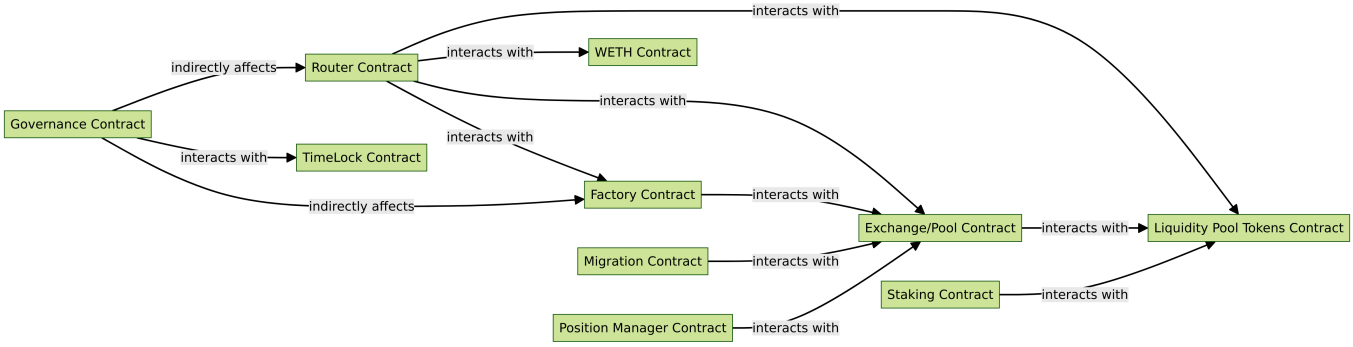

Fig. 1. Network representation of the dApp

- The **Liquidity Pool Tokens Contract** issues LP tokens, which represent a user's share in the liquidity pool and interacts with the Exchange/Pool Contract.
- The **Governance Contract** indirectly affects the Router and Factory Contracts, as it governs protocol changes that may alter their behavior.
- The **TimeLock Contract** interacts with the Governance Contract to ensure that changes are implemented with a delay for community review.
- The **Staking Contract** and **Position Manager Contract** interact with the Exchange/Pool Contract, managing staking rewards and concentrated liquidity positions, respectively.
- The **Migration Contract** enables the transfer of liquidity from one protocol version to another, indicating the system's upgradability.

Highlighted in the network is the interaction with the **WETH Contract**, which is external to the dApp, illustrating the dApp's reliance on external components for essential operations like trading ETH. This reliance exposes the dApp to risks associated with those contracts, such as vulnerabilities or downtime in the WETH Contract potentially impacting the dApp's ability to process ETH trades.

The graph demonstrates the modular and interdependent architecture of the dApp, allowing for an analysis of how the system responds to changes and external events. For instance, if an external event caused a surge in interactions with the WETH Contract, it could create bottlenecks impacting the Router Contract's efficiency. Studying a dApp as a complex system [1], [2] provides insight into how dApps maintain robustness while revealing structural vulnerabilities related to their network.

By considering various levels of analysis granularity—such as contract level, function level, and the use of bipartite graphs for analysing contract-function or function-contract relationships—we can uncover insights into potential weaknesses and how they might be exploited or mitigated.

A. Contract-level

At the contract level, we examine the network of smart contracts and how they interact with one another. The analysis explores the network of smart contracts that constitute the backbone of a dApp, focusing on how these contracts, as distinct entities, interact and form the foundation of the application. Our example involves different contracts—such as the Factory, Exchange/Pool, Router, LP Tokens, and Governance contracts—each fulfilling specific roles within the ecosystem.

The Factory Contract acts as a central point for creating new Exchange/Pool Contracts, establishing a dynamic marketplace for various cryptocurrency pairs. The Router Contract serves as the interface for users, directing trades and liquidity provisioning to the appropriate Exchange/Pool Contracts. This modular setup allows for efficient management of trading and liquidity across different token pairs.

Intercontract interactions are key to the dApp's operation, with each contract depending on others to perform its functions. For example, the Router Contract relies on the Factory Contract to locate or create the necessary Exchange/Pool Contract for a user's trade request. Similarly, the Governance Contract plays an important role in facilitating protocol upgrades and changes, influencing the behavior and parameters of other contracts within the ecosystem.

Understanding the contract-level structure is extremely important for identifying potential systemic risks and points of failure. The Router Contract's central role in facilitating trades and liquidity means that any compromise in its operation could have impacts on the dApp's overall functionality. Similarly, the Governance Contract's ability to enact changes across the platform highlights the importance of secure and robust voting mechanisms to prevent malicious or unintended alterations to the system.

By examining the network at the contract level, we can gain insights into the dApp's architectural design and identify critical components that ensure its stability and security. This view allows us to better understand the interdependencies among contracts and the importance of each in maintaining the dApp's ecosystem.

B. Function-level

Function-level analysis focuses on the interactions between individual functions across different contracts. This granularity

helps identify specific points of failure that might not be apparent at the contract level.

Let's consider a function in the Liquidity Pool Tokens Contract that issues LP tokens based on the liquidity provided. The function's logic might involve complex calculations to determine the amount of LP tokens to issue based on the current state of the liquidity pool. Suppose this function contains a rounding error when calculating the user's share or fails to properly handle integer overflows (for example, when large amounts of liquidity are added to the pool). In such cases, an attacker could deliberately create transactions that exploit these vulnerabilities to either receive more LP tokens than they are entitled to or disrupt the token distribution mechanism, affecting the fairness and integrity of the liquidity pool.

In the Uniswap-like dApp, the Exchange/Pool Contract and the Liquidity Pool Tokens Contract work closely together. The Exchange/Pool Contract handles the addition of liquidity to the pool and interacts with the Liquidity Pool Tokens Contract to issue LP tokens. A potential overflow vulnerability in the "addLiquidity" function of the Exchange/Pool Contract could, for example, pass incorrect parameters to the LP Tokens Contract, leading to incorrect LP token issuance.

To prevent overflow and underflow vulnerabilities, the smart contracts should use the SafeMath library (or similar mechanisms which include built-in overflow checks) for all arithmetic operations. This ensures that all calculations are performed safely without unexpected wraparounds.

To mitigate reentrancy risks associated with external calls made by functions, adhering to the Checks-Effects-Interactions design pattern ensures that state changes are finalized before any external interactions occur. This pattern can be crucial in functions that involve token transfers or interactions with external contracts. By zooming in on the function-level interactions and vulnerabilities within the smart contracts of the Uniswap-like dApp, developers can gain a detailed understanding of potential security risks.

C. Bipartite graphs

Bipartite graphs offer a way to analyse the relationships between two distinct sets of nodes. In this context would consist of two distinct sets of nodes: one representing the smart contracts (such as the Router Contract, WETH Contract, Exchange/Pool Contract, etc.) and the other representing the functions within these contracts (e.g., addLiquidity, removeLiquidity, swapETHForToken, etc.). Edges in this graph would connect contracts to their respective functions, illustrating the operational flow and dependencies within the dApp.

In this direction, Ibba et al. developed MindTheDapp [5], a novel toolchain for analysing Ethereum-based dApps through a complex network-driven approach, transforming smart contract interactions into a specialized bipartite graph for advanced network analytics.

For instance, the function for wrapping ETH into WETH (wrapETH) is key for enabling ETH trades against ERC-20 tokens in the dApp. This function, located within the WETH Contract, is used for operations involving ETH, as it allows ETH to be treated as an ERC-20 token, maintaining consistency within the dApp's trading mechanism. Similarly, the addLiquidity function in the Exchange/Pool Contract is central for maintaining the liquidity pools' health.

A bipartite graph analysis could reveal that these key functions (wrapETH, addLiquidity) are heavily reliant on interactions with a limited number of other contracts—highlighting a concentration of dependency that could pose risks if those contracts are compromised. For example, if the WETH Contract were susceptible to a reentrancy attack, it could jeopardize not just the wrapping/unwrapping of ETH but also ripple through the ecosystem, affecting trades and liquidity provisions that depend on the integrity of wrapped ETH.

Understanding the critical pathways and dependencies allows for implementing strict security measures around key functions. For functions identified as high-risk due to their centrality or dependency concentration, integrating circuit breakers can provide a fail-safe. These mechanisms allow for the temporary halting of contract functions in response to detected anomalies or attacks, mitigating potential damage while a permanent fix is developed. Designing contracts with upgradability in mind ensures that vulnerabilities identified through bipartite graph analysis can be addressed without overhauling the entire system. Proxy patterns and upgradeable smart contract designs allow for the selective updating of critical functions or contracts that present elevated risks. Implementing a governance model for critical updates or changes to high-centrality functions and contracts can ensure that modifications are made transparently and with community consensus. This reduces the risk of unilateral changes that could introduce vulnerabilities or disrupt the dApp's operations. By employing bipartite graph analysis, developers and auditors can gain a complete view of the operational and functional dependencies. This detailed mapping facilitates a targeted approach to securing the ecosystem, ensuring that critical functions and their supporting contracts are fortified against potential threats, enhancing the overall robustness and reliability of the decentralised application.

V. Smart Contract Software Metrics

Incorporating additional metrics into the complex network analysis of dApps' smart contracts and functions can significantly enhance our understanding and evaluation of the system's robustness, efficiency, and security [4], [8], [10]. Traditional software metrics, alongside newly defined metrics specific to the blockchain and smart contract domain, provide a multidimensional view of each node (smart contract or function) within the network graph. These metrics could include code complexity, gas usage, transaction throughput, reusability, and the frequency of function calls, among others. By assigning these metrics to nodes, we gain a richer dataset for analysis, allowing for a deep assessment of the system's behavior under various conditions.

Code complexity metrics, such as cyclomatic complexity, can give insights into how intricate a smart contract or function is, potentially indicating harder-to-maintain code or higher

susceptibility to bugs. Gas usage metrics for functions can highlight areas where optimisations are necessary to reduce transaction costs and improve the system's scalability. Measuring transaction throughput for each contract or function can identify bottlenecks in the system, where performance might degrade under high demand. Reusability metrics, which assess how often code is reused across the system, can indicate a healthy modular design or, conversely, areas where too much dependency on a single function or contract could pose risks.

The frequency of function calls is a dynamic metric that reflects how users interact with the dApp. High-frequency nodes might be critical paths that require additional security measures or optimisations to ensure the system's reliability and responsiveness. Newly defined metrics for smart contracts, such as upgradeability score (how easily a contract can be upgraded to fix vulnerabilities), decentralisation degree (the extent to which control and decision-making are spread across the network), and inter-contract coupling (the degree of dependency between contracts), can offer insights into the architectural and operational qualities of the dApp. These metrics can help identify strengths and weaknesses in the system's design and governance.

By integrating these metrics into the complex network analysis, researchers and developers can perform analyses at both the smart contract and function levels. For instance, analysing the correlation between code complexity and gas usage across nodes can identify areas for optimisation. Incorporating additional metrics into the network graph transforms it from a static representation of relationships into a dynamic, informative tool that can guide decision-making processes, from code optimisations to architectural adjustments. This approach to analysing dApps not only aids in identifying current issues but also in predicting future challenges, enabling proactive measures to ensure the system's robustness, efficiency, and security.

VI. Conclusion

In this position paper, we have examined how complex systems theory can be applied to the study of decentralised applications (dApps). This exploration stressed the significance of grasping the detailed interactions and dependencies within the networks of smart contracts fundamental to these platforms. By applying concepts from complex systems, we can increase our understanding of dApps' behavior, vulnerabilities, and potential areas for optimisation [3], [6], [7].

Our discussion began with an introduction to complex systems theory and its relevance to dApps, highlighting how this perspective allows for a deeper comprehension of their functionality and inherent challenges. We then provided an overview of the architecture of a Uniswap-like dApp, noting the roles of various smart contracts and how they interact within this modular and interconnected framework. A specific use case was explored to illustrate the dynamic between users, smart contracts, and the Ethereum blockchain, emphasising the transaction flow and the critical role each contract plays in facilitating exchanges and liquidity provision.

Further suggestions included modeling the dApp's smart contracts as nodes within a complex network graph, which could shed light on the structural relationships and potential vulnerabilities within the system. This approach can help pinpoint critical components and assess their influence on the system's stability and security. It is possible to achieve additional depth by analysing functions within contracts through function-level analysis and employing bipartite graphs, which can allow for a focused examination of vulnerabilities and dependencies at a more granular level.

The integration of traditional software metrics and newly defined ones can further enrich the complex network analysis, providing a complete view of each node's performance, security, and efficiency.

Applying complex systems theory to the analysis of decentralised applications lays out a thorough framework for understanding their architecture, functionality, and vulnerabilities. The methodologies discussed enable a detailed exploration of the interactions between smart contracts and their functions. This strategy aids in identifying potential weaknesses and security threats while also identifying opportunities for improvement and innovation.

References

[1] Sabrina Aufiero, Giacomo Ibba, Silvia Bartolucci, Giuseppe Destefanis, Rumyana Neykova, and Marco Ortu. The network structure of smart contracts in ethereum dapps. *Complex Networks 2023 (to appear)*, 2023.

[2] Sabrina Aufiero, Giacomo Ibba, Silvia Bartolucci, Giuseppe Destefanis, Rumyana Neykova, and Marco Ortu. Dapps ecosystems: Mapping the network structure of smart contract interactions. *arXiv preprint arXiv:2401.01991*, 2024.

[3] Lexi Brent, Neville Grech, Sifis Lagouvardos, Bernhard Scholz, and Yannis Smaragdakis. Ethainter: a smart contract security analyzer for composite vulnerabilities. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 454–469, 2020.

[4] G Ibba, S Khullar, E Tesfai, R Neykova, S Aufiero, M Ortu, S Bartolucci, and G Destefanis. A preliminary analysis of software metrics in decentralised applications. In *Proceedings of the Fifth ACM International Workshop on Blockchain-enabled Networked Sensor Systems*, pages 27–33. ACM, 2023.

[5] Giacomo Ibba, Sabrina Aufiero, Silvia Bartolucci, Rumyana Neykova, Marco Ortu, Roberto Tonelli, and Giuseppe Destefanis. Mindthedapp: a toolchain for complex network-driven structural analysis of ethereum-based decentralised applications. *arXiv preprint arXiv:2310.02408*, 2023.

[6] Giacomo Ibba and Marco Ortu. Analysis of the relationship between smart contracts' categories and vulnerabilities. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1212–1218. IEEE, 2022.

[7] Giacomo Ibba, Giuseppe Antonio Pierro, and Marco Di Francesco. Evaluating machine-learning techniques for detecting smart ponzi schemes. In *2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 34–40. IEEE, 2021.

[8] Marco Ortu, Matteo Orrú, and Giuseppe Destefanis. On comparing software quality metrics of traditional vs blockchain-oriented software: An empirical study. In *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 32–37. IEEE, 2019.

[9] Giorgio Parisi. *In un volo di storni*. Rizzoli, 2021.

[10] Roberto Tonelli, Giuseppe Antonio Pierro, Marco Ortu, and Giuseppe Destefanis. Smart contracts software metrics: A first study. *Plos one*, 18(4):e0281043, 2023.