Mining Domain Models in Ethereum DApps using Code Cloning

NOAMA FATIMA SAMREEN, Ryerson University, Canada
MANAR H. ALALFI, Ryerson University, Canada

This paper discusses and demonstrates the use of near-miss clone detection to support the characterization of domain models of smart contracts for each of the popular domains in which smart contracts are being rapidly adopted. In this paper, we leverage the code clone detection techniques to detect similarities in functions of the smart contracts deployed onto the Ethereum blockchain network. We analyze the clusters of code clones and the semantics of the code fragments in the clusters in an attempt to categorize them and discover the structural models of the patterns in code clones.

CCS Concepts: • Software and its engineering → Integrated and visual development environments; Software libraries and repositories; System modeling languages; Unified Modeling Language (UML); • Computing methodologies → Modeling methodologies.

Additional Key Words and Phrases: Blockchain Technology, Ethereum Smart Contracts, Code Cloning, Model-Driven Engineering

ACM Reference Format:
Noama Fatima Samreen and Manar H. Alalfi. 2018. Mining Domain Models in Ethereum DApps using Code Cloning. 1, 1 (March 2018), 19 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Model-driven engineering (MDE) make use of structural and behavioral models that can be used to help a novice programmer of smart contracts understand the new features provided by the blockchain-specific programming languages such as Solidity [1]. These models provide a way of abstracting technical and language-dependent functionalities thereby increasing the structural and behavioral understanding of a new technology such as the blockchain technology. Given the stigma around the complexity of development of decentralised applications (DApps) using smart contracts, model-driven development could enable the widespread adoption of the blockchain technology as software developers of other technologies could use this as a starting point of their blockchain oriented software engineering process. MDE is a strategic framework that is used to realize advanced solutions by considering different aspects and stakeholders involved in a domain. By viewing different models we can achieve separation of concerns with a higher level of abstraction and reduces the complexity of dealing with DApps specification.

Code clone detection in software systems has many applications. For example, it can be used to identify repeated similar code fragments with the aim of developing libraries and standards. It can also be used to enable bug fixes, updates and changes to a software system [2].

In this paper, we leverage code clone detection technique by using Nicad [3] to detect similarities in functions of the smart contracts deployed onto the Ethereum blockchain network. We analyse the clusters of code clones and the
semantics of the code fragments in the clusters in an attempt to categorise them and discover the structural models of the patterns in code clones. These structural models of patterns produced by code clone detection technique could then be used as conceptual models in defining low-level semantic Meta Models. These structural domain models are discovered by reverse engineering of the largest code clone clusters in the source code files of smart contracts deployed on the Ethereum blockchain network.

Our aim in this research is to identify patterns of code reuse through code clone detection technique to build models on top of the existing code patterns in Ethereum smart contracts. We approach this study by investigating the extent of code clones in Ethereum smart contracts, categorizing these clone clusters and producing structural domain models from these clusters to enable code reuse at the modelling level. We highlight that our research leverages an existing code clone detection tool, NiCad that operates at the source code level in order to detect clones of type 1, 2, 2-c, 3 and 3-c. In this paper, we address the following research questions:

**RQ1:** How can we identify and analyse the code clones present in main-net verified smart contracts that are associated with DApps?

**RQ2:** Can the clone clusters of smart contracts be categorised into domains depending on their functionality?

**RQ3:** Can the categorised clone clusters of smart contracts be reverse engineered to produce structural domain models?

Our workflow for discovering structural domain models from Ethereum smart contracts consists of following steps:

1. **Smart contracts code usage analysis:** Identification of highly used code patterns by leveraging code clone detection technique. As BT and smart contracts are just being adopted in various industries, we initiate our models discovery process by analyzing the extent and characterization of code clones in already deployed smart contracts.

2. **Extraction of Domain Models (Code-to-Model Transformation):** We realise this step by reverse engineering the highly cloned code patterns identified and semantically categorised in our code clone detection phase.

3. **Defining Meta Model from structural domains:** Each structural domain is then characterized by an ontological basis, which describes the structural domains of the concepts and the relations between them to define a Meta Model for a DApp system (see 10). Besides this description of the structure of the domains, the meta model could
also provide semantics for the concepts used. This meta model would capture the complete vocabulary of the transactions necessary to develop a smart contract based decentralised application (DApp).

2 BACKGROUND

2.1 Ethereum Smart Contracts and Solidity

Blockchain oriented software engineering to develop Decentralised applications (DApps) that execute smart contracts is complicated. Ethereum Smart Contracts [4] are typically written in a high-level turing-complete programming language such as Solidity [1], and then compiled to the Ethereum Virtual Machine (EVM) byte-code [4], a low-level stack-based language. These smart contracts cannot be updated once deployed to the main-net. This requires the developers to ensure that these smart contracts are attacks-resistant and reflect complete business logic desired by the DApp developers. A large portion of the problems with smart contract development could be pinpointed to the lack of formalization of blockchain oriented software engineering. The lack of standards and best practices makes smart contract development prone to problematic practices.

2.2 Code-Cloning

Reusing code for similar applications by cloning is an efficient way of minimizing the cost and time of software development. Research shows that the use of templates in software programming results in various advantages. It is especially very beneficial when there is a lack of knowledge base of a newly developed programming language. Developing templates from code clones and establishing them as standards or libraries [5] also affects a program by reducing its size making it easier to read and comprehend.

2.2.1 Types of Code Clones. Code clones classification used in this analysis is given in Table [1].
Table 1. Classification Of Code Clones

| Clone Type | Clone Type Definition | Description |
|------------|-----------------------|-------------|
| 1          | Exact                 | Code clones of Type-1 are code fragments that are identical with variation in comments and blank spaces. |
| 2          | Renamed clone - Blind Renamed | Code clones of Type-2 are code fragments that are identical with variations in identifier names, literals, comments and blank spaces. Blind Renaming is the process of renaming all the identifiers to \( X \) without keeping a record of previously renamed identifier. |
| 2c         | Renamed clone - Consistently Renamed | Code clones of Type-2c are similar to the Type 2 code clones except that they are consistently renamed, which is process of renaming the identifiers to \( X^n \). Where, \( n \) is the number of occurrence of the identifier with respect to the previously renamed identifier. |
| 3-1        | Near miss clone - Blind Renamed | Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, white-space, layout and comments. |
| 3-2c       | Near miss clone - Consistently Renamed | Code clones of Type-3-2c can be considered as a hybrid of the Type 3-1 and Type 2c code clones in that they are similar code fragments with changed, added or removed statements, in addition to variations in identifiers, literals, types that are consistently renamed. |
| 4          | Functional Clone      | Two or more code fragments that perform the same computation but are implemented by different syntactic variants. |

2.2.2 NiCad Code Clone Detector. The NiCad clone detection tool is an efficient near-miss code clone detection tool [3]. In this paper, we demonstrate the detection of code cloning in Ethereum Smart Contracts using a static clone detection method. Our hypothesis is that near-miss clone detection will provide a means of implementing a model-driven development framework by producing a domain models of smart contracts for each of the popular domains in which smart contracts are being rapidly adopted.

2.3 Model-Driven Development for DApps

The model-centric or model-driven development approach focuses on utilization of models to describe the structure and the behavior of a system, which is then used to generate source code for a system. It can lower the complexity and create a better understanding of the system. [6]. To automate and make more powerful maintainable smart contracts, we need to extend the MDE techniques to design and develop complex and decentralized applications. Smart contracts and DApps build on the premise of having a tremendous impact not only on the finance domain but also in other domains like gaming, supply-chain etc. However, its adoption requires novel skill sets that current software professional profiles
fail to meet despite the increasing demand. Developing DApps using smart contracts would be easier if the learning
curve for the needed skills would be more convenient. Model-driven software engineering can help in abstracting
blockchain technology and, starting from these abstractions, enabling automated smart contract code generation.

3 APPROACH

Our approach in producing domain models from existing smart contracts deployed on the Ethereum blockchain network
is described in Figure 2. It consists of following steps:

3.1 Dataset

Our data-set consisted of 10680 smart contracts that run behind a DApp. These smart contracts were collected from
Etherscan [7], a web portal that provides analytical data and repository of smart contracts deployed onto the Ethereum
blockchain network. Etherscan provides verification of smart contracts and meta-data about a smart contract and its
owner address. Partial meta-data for analysis of domain deduction was even collected from various literature surveys
available online in the form of token white-papers, and GitHub repositories of contracts creators corresponding to a
representative smart contract of a code clone cluster.

Demographics of the Analysed smart contracts

To provide a proper context for the results, this section describes the demographics of the smart contracts in our
dataset. Our dataset of total 10680 sol files collected from Etherscan consisted of sub-contracts, libraries, interfaces,
events, modifiers. To understand the models and the metamodel derived from our dataset, we provide the statistics of
analysed smart contracts. As shown in the Table 2, we analyzed 49,128 smart contracts which included main contract
and its depended contracts. Other factors analysed included 8780 libraries, 2901 interfaces, 42101 events, 2901 modifiers.
The extent of division of functionality into smaller sub-contracts is practised by software developers to achieve reduced
complexity of contracts. Besides, the events information shows that the execution information will be logged in
Ethereum. The interfaces, libraries, modifiers and events information is used to derive structural models and templates.

The final domain categorization of code clone clusters is provided in the following steps.

3.2 Using NiCad Near-Miss Code Clone Detection Tool

The most recent version of NiCad code clone detector (v6.2) has built-in support for parsing and analysing source
code files written in the Solidity programming language. However, the tool was not able to parse around 2800 smart
contracts because of version incompatibility of the Solidity parser provided by the Nicad tool. We therefore updated the
Solidity grammar to the latest version of the Solidity programming language to be able to parse and analyse all the
smart contracts in our data-set.

3.3 Domain Categorization of Large Code Clone Clusters

The main step of our research was to deduce domain categories based on the metrics of the code clone clusters produced
by Nicad tool and even factoring in the semantics of code represented by these clusters. We leveraged the meta-data

| Total Sol Files | Contracts | Libraries | Interfaces | Events | Modifiers |
|----------------|-----------|-----------|------------|--------|-----------|
| 10680          | 49128     | 8780      | 2901       | 42101  | 29014     |
Table 3. Code Clones Found in a Dataset of 10680 Smart Contracts Collected From Etherscan [7]

| Clone Type | Extractor Only | Filtered and Blind-Renamed | Filtered Consistent-Renamed |
|------------|----------------|---------------------------|---------------------------|
| Clone Type | Type 1 | Type 3-1 | Type 2 | Type 3-2 | Type 2c | Type 3-2c |
| Clone Pairs | 17012 | 29645 | 18534 | 35115 | 18523 | 21487 |
| Clone Classes | 596 | 607 | 753 | 750 | 752 | 762 |
| Max diff threshold | 0% | 30% | 0% | 30% | 0% | 30% |

Table 4. A Summary of the Categorization of the Top Clusters of Code Clones in a Dataset OF 10680 Ethereum Smart Contracts

| S.no | Cluster Domain Category | Cluster Size | Accumulated Cluster Similarity Percentage (Min/Max) | Domain Category deduced by clone detection type |
|------|-------------------------|--------------|-----------------------------------------------------|-------------------------------------------------|
| 1    | Token Management        | 547          | 90%/100%                                            | Type 1,2,2c,3-1,3-2c                              |
| 2    | Arithmetic Operations   | 309          | 91%/100%                                            | Type 1,2,2c,3-1,3-2c                              |
| 3    | Exchanges               | 214          | 70%/100%                                            | Type 2c, 3-1, 3-2c                                |
| 4    | Finance                 | 70           | 75%/85%                                             | Type 3-2c                                        |
| 5    | Data Oracles            | 62           | 70%/77%                                             | Type 3-2c                                        |
| 6    | Marketplace             | 53           | 72%/85%                                             | Type 3-2c                                        |
| 7    | Gaming                  | 42           | 77%/98%                                             | Type 2c, 3-2c                                    |
| 8    | Security                | 41           | 70%/77%                                             | Type 2c, 3-2c                                    |

collected from various resources such as Etherscan [7], StateofTheDApps [8] and DAppRadar [9] regarding the smart contracts in largest clusters to infer upon the domain categories. The final list of our domain categories was a result of an extensive manual browsing through the clusters to understand the semantics of the code fragments in the clusters and extracting relevant meta-data about the code fragments from online resources.

3.4 Reverse Engineering of Domain Categories To Produce Domain Models

After we formulated the list of domain categories exhibited by the largest code clone clusters in our analysis, we reverse engineered a representative smart contract of the cluster with an ultimate goal of producing structural models of the domain categories. The reverse engineering step was performed using the [10] tool available as an extension to the web-based Remix IDE for Solidity.

3.5 Defining Meta Model

The reverse engineered structural domain models showed certain overlap of patterns between domains. We therefore, aim at defining a Meta Model that represents common metadata required for deep semantic extraction of business rules.
and transformation of domain models. The defined Meta Model describes the structural domain models of the concepts and the relations between them of a DApp system (see Figure 10).

4 RESEARCH FINDINGS

RQ1: Code Clones in Ethereum smart contracts - Analysis

Motivation. First, we aim at identifying and characterizing the code clones in Ethereum smart contracts that are associated with DApps to be able to understand the diversity of use-cases implemented by Ethereum smart contracts.

Discussion. To answer our RQ1, we analyzed the output of NiCad near-miss clone detection tool. Some of the terms to understand before analysing the code clone detection report are:

(1) Code Granularity: It refers to the level of structural source unit taken into consideration while analysing a source code file for clones. In this paper, we configure NiCad clone detection tool to the functions granularity to detect code clones in the functions of different Solidity smart contract code files.

(2) Clone Pair: Two code fragments which are identical or similar are known to have a clone relation between them and are called a clone pair.

(3) Clone Class: It is the maximum set of code fragments which contains a clone pair for each code fragment.

(4) Max. Difference Threshold: It is the percentage of difference in code fragments metric to be configured in the NiCad code clone detection tool to allow detection of code clones with a difference of up to the specified threshold.

We configured it to allow code clones with a maximum difference threshold of up to 30%.

The analysis results of a data-set of smart contracts are given in Table3. We were able to detect code clones of Types 1, 2, 2c, 3-1, 3-2c using the NiCad near-miss code clone detection tool. A total of 40502 functions were extracted from the source code files and the clone size varied from 10 to 2500 lines of code.
Observation 1: One of the observations we made while analysing code clone reports generated by NiCad is that even though there is a widespread adoption of Ethereum blockchain technology and smart contract, the source code files exhibit very similar functionalities. The majority of the smart contracts are either token contracts or token management programs. It can be inferred that the smart contract developers are mostly cloning code intentionally from the established standards available online (e.g., OpenZeppelin Libraries [11]) to avoid any unforeseen security vulnerabilities that may arise because of the lack of knowledge and experience in this new technology.

Observation 2: Another noteworthy observation was that over 500 clone classes of smart contracts were code clone pairs arising from non-standard contracts. We believe that this is because of the immutability characteristic of the blockchain technology that forces a smart contract developer to deploy the same contract again even with minor changes, such as updating the compatibility of their smart contracts to the latest version of the Solidity programming language.

RQ2: Code Clones in Ethereum smart contracts - Categorization

Motivation. To extract domain models for an ultimate goal of extracting business rules from existing Ethereum based decentralised applications (DApps), we categorize the largest code clone clusters identified in the code clone detection of Ethereum smart contracts.

Discussion. We approached the categorization of our code clone clusters depending on the cluster similarity percentages. Specifically, we categorised the code clone clusters into a domain category if a unique clone cluster has a similarity percentage of 70% or higher. However, after a manual analysis of the clusters we came across multiple clusters of varying similarity percentages that could be categorised into a single domain category. To overcome this redundancy.

Manuscript submitted to ACM
of clones and deduce one domain category of related functionalities, we analysed the source code of a representative smart contract in each of the clusters and accumulated the code clone clusters into a single domain category of related functionalities. Our approach resulted in code clone clusters being categorised into following domain categories. [See Table 4]:

1. **Token Management**: The largest code clone cluster was related to smart contracts that handled tokens. Most of the clusters were an implementation of the well-known token management standards such as ERC-20 \(^1\) (Token management standard for fungible tokens) and ERC-721 (Token management standard for non-fungible tokens). Almost all the other domain categories deduced by our analysis leveraged this cluster of code clones to be implemented along with their specific functionalities.

2. **Arithmetic Operations**: Earlier versions of the Solidity programming language was detected with certain arithmetic vulnerabilities arising from it syntax. Before the release of the latest version of the Solidity programming language, the smart contract developers adopted the practice of implementing the *SafeMath* library provided online to safely perform the arithmetic operations in a smart contract. This practice is evident from our analysis and the second largest cluster of code clones contributed to the implementation of *SafeMath* library. Although the latest version of the Solidity programming language has this vulnerability resolved, our data-set consisted of smart contracts ranging from the versions 0.5.0 to 0.8.0. [1]

3. **Exchanges**: The exchanges domain category consisted of use-cases involving trading of digital assets built upon the Ethereum blockchain technology. The functionalities consisted but not limited to minting, burning and swapping of digital assets like fungible and non-fungible tokens.

   **Observation 3**: We observed that the popular cryptocurrency exchange names like [12], [13], [14] were listed in this cluster. One of the surprising notes was that some of the above mentioned established and popular exchange platforms had a similarity percentage of up to 100% in this cluster. This information could be used to infer upon the extent of plagiarism or inadequate copyright policies in the world of blockchain technology based decentralised applications.

   **Observation 4**: 60% of the smart contracts in this cluster also leveraged the *Data Oracles* code clone cluster discussed later in this section. It was also one of the clusters that had smart contracts leveraging *Token Management* code clones extensively.

4. **Finance**: One of the industries that has the most potential of getting revolutionized by the Ethereum blockchain technology is the Finance industry. DeFi (Decentralised Finance) refers to financial services like accessing and handling financial assets in a universal and decentralized manner using the blockchain technology.

   **Observation 5**: Despite all the hype around the DeFi solutions provided by the Ethereum blockchain technology and smart contracts, our code clone analysis showed that the smart contracts developed behind the DeFi DApps were very limited and homogeneous when compared to the use-cases provided by traditional financial applications. The *DeFiLendingPool* smart contract had a similarity percentage of up to 71% that consisted of use-cases revolving around lending, borrowing, withdrawing and depositing of assets. It leveraged a form of *Token Management* cluster of code clones as well.

5. **Security**: Blockchain technology is based on cryptographic signatures to authenticate an owner of an address without disclosing its private key. These signatures are primarily used for signing transactions and sometimes also to sign arbitrary messages. Ethereum blockchain technology uses the Elliptic Curve Digital Signature

---

\(^1\)ERC: Ethereum Request for Comments, Common ERC standards define a required set of functions for a token type [5]
Algorithm [15]. ECDSA is only a signature algorithm that cannot be used for encryption and consists of two integers: \( r \) and \( s \). Ethereum uses a recovery identifier \( v \), thereby making the signature as \( \{r, s, v\} \). These message signatures can be used to verify the ECDSA signatures using smart contracts. A function to recover the address of the private key of the message signatory uses a pre-compiled contract at address 0x1. Signature verification in smart contracts finds many useful applications such as, multi-sig contracts and exchanges. The code clone
cluster representing the ECDSA message signatures validation functionalities implemented a form of standard ERC-1271 provided by the Ethereum blockchain technology.

(6) Marketplace: Blockchain technology has encompassed $250 million market of NFTs (non-fungible tokens) that are traded as digital assets. NFTs include artefacts ranging from digital trading of art to virtual real estate and gaming collectibles. A fungible token (FT) can be directly traded with another FT, however, same cannot be said about NFTs. These marketplace DApps that allow trading of NFTs ranged anywhere from $10 to hundreds of thousands of dollars and yet lack security and proper copyright validation checks.

**Observation 6:** While analyzing the code clone clusters representing trading of NFTs in smart contracts, we made an interesting observation that the domain categories are not clearly separated in the Ethereum DApps ecosystem. For example, the DApps *AxieInfinity* [16] and *Decentraland* [17] contained code fragments from Marketplace domain category and they also exhibited strong code functionalities categorised under Gaming domain category.

(7) Data Oracles: The blockchain technology is somewhat a closed network with complexity revolving around extracting data from the blockchain network and access data from external sources into the blockchain network. However, with blockchain based DApps receiving growing interest from different industrial domains, it is necessary to enable a smart contract to access external data required to control the execution of the business logic. Data oracles provide an efficient interface to access external data from various data oracles in to a smart contract. These oracles can be modified to cater for different types of data depending on the industry and requirements. *Oraclize or Provable Things* [18] and *ChainLink* [19] are the most widely used data oracles in the blockchain...
network. A data request to these data oracles is validated against the inclusion of the following information from the blockchain network: a data source type, a query, and an optional authenticity proof.
Mining Domain Models in Ethereum DApps using Code Cloning

(8) Gaming: Gaming DApps in blockchain networks have gained popularity because of the involvement of trading of financial/digital assets in the form of NFTs and sometimes even FTs while playing the game.

**Observation 7:** We observed clusters of code clones depicting a similarity percentage of 100% of successful and popular Ethereum based gaming DApps such as Cryptokitties [20] and AxieInfinity [16]. Given the fact that there are not many strict copyright protection systems in place for blockchain DApps, there is an evident plagiarism of gaming DApps by the DApps developers to benefit financially.

RQ3: Code Clones in Ethereum smart contracts - Modelling

**Motivation.** To be able to re-use the existing models in the Ethereum smart contracts and modernize them to integrate newer business rules to diversify the use-cases of Ethereum based DApps, we reverse engineer the extracted code clones from Ethereum smart contracts to produce domain models.

**Discussion.** Figures {3, 4, 5, 6, 7, 8, 9} show the structural models produced by a manual reverse engineering of code fragments from the largest code clone clusters of smart contracts categorised into domains in the above step.

The **Token Management** domain category mainly represents the ERC20 and ERC721 [5] token standards and therefore the domain model extracted from reverse engineering the code clone cluster representing the functionalities mostly cloned by smart contracts is showed in Figure 3. This model lists functions related to management of tokens in a DApp, such as transfer, checking balance, setting allowance of transfer, approval of transfer etc.

Similarly the **Arithmetic Operations** domain category uses the SafeMath (see Figure3) interface to represent the operations that can be performed with the surety of protecting the smart contract from the vulnerability that existed in the Solidity programming language regarding Integer Overflow.

The **Exchanges** domain category model (see Figure5) represents the functions that were cloned the most by the currently popular cryptocurrency exchange platforms available. The functionalities represented in this model deal with minting, burning and swapping of digital assets and currencies.

The **DeFi** domain category model (see Figure6) was reverse engineered from code clone clusters that required more semantic analysis of the code than the rest of the domain categories. The **DeFiLendingPool** smart contract represented most of the functions that were cloned by other DeFi applications, which consisted of use-cases related to lending, borrowing, depositing and withdrawing of digital assets from a smart contract defined lending pool.

The **Data Oracles** domain category uses the Provable data oracle available online to access external data into a smart contract for efficiently executing a business logic.

The domain model depicts the Oracle and Oraclize Abstract contracts that easily integrate the external data into a blockchain application. (see Figure8)

Trading of digital assets, primarily NFTs, is realised by the **Marketplace** domain category. The functions related to the business logic of this domain consisted of creating and cancelling of auctions, maintaining order history, order management such as receiving, executing, filling and cancelling a order. It also leveraged the **Token Management** standard of ERC721 to manage the exchanges of NFTs. (see Figure4)

The **Gaming** domain model represents the largest code clone clusters related to a smart contract behind a Ethereum based gaming DApp. However, the model extracted from this cluster is generic and does not relate to the business rules of the game specifically. The model shows the structure of smart contracts to manage ownership of the game, mint new tokens and get new collectibles in the game. (see Figure7)

The **Security** model represents the signature validation structure implemented by smart contracts using the ECDSA algorithm to sign the transfers and arbitrary messages. (see Figure9)
Fig. 10. Meta Model for the mined domain models
5 RELATED WORK

Code Cloning in Ethereum Smart Contracts. In the space of code cloning analysis and detection tools for Ethereum smart contracts, there are only a few studies so far. The most similar study to ours is the work of Kondo et al. [21] who study code cloning in Ethereum at the source code level with a goal to quantify the number of clones in Ethereum and their characteristics. They follow the ideology of positive impact of code cloning in software systems to avoid taking the risk of writing vulnerable code. The authors compare the code clone clusters to an online library of publicly available open-source smart contracts on OpenZeppelin [11]. They analyse the code cloning in a data-set of 33,073 smart contracts and resulted in a clone ratio of around 80%. [21] employs a tree-based clone detector, Deckard [22] at the source code level and analyse the detected code clones to understand the Ethereum blockchain technology ecosystem. We agree with the authors observation that although the blockchain technology is gaining adopting in a wide range of use cases, currently there is a lack of diversity in its applications.

Another research study by Gao et al. produced a code clone detection tool for Ethereum Smart Contracts called SmartEmbed. This tool identifies code clones from the source code of smart contracts using the code embedding and similarity checking techniques. SmartEmbed is aimed at quantifying clones in Ethereum Smart Contracts and identify the clone-related vulnerabilities. SmartEmbed was evaluated on a data-set of around 22,000 smart contracts contracts and resulted that the clone ratio in Ethereum Smart Contracts is close to 90%. [23]

Other than the above mentioned two studies, some researchers also analysed the code cloning in Ethereum Smart Contracts at the byte-code level.

A research study by Liu et al. that detects code clones in Ethereum Smart Contracts using the byte-code generated by the Ethereum Virtual Machine (EVM) of each deployed smart contract. is more challenging and less advantageous towards derivation of models by reverse engineering of the code clone as the byte-code factors in the version number of the Solidity programming language being used and could potentially miss some of the core functional code clones across smart contracts. [24]

We expand upon these papers by investigating code clones of type 3, which focuses on near miss renamed code clones rather than just the code clones resulting from identical code reuse. Moreover, we contribute to the research on code re-use in Ethereum smart contracts by providing a categorization of these investigated large clusters of code clones. These clusters are further subjected to reverse engineering to produce domain models to facilitate code reuse at the model level.

Model-Driven Development and Ethereum based decentralised applications (DApps) development. A Model-Driven Reverse Engineering (MDRE) approach proposed in [25] aims at extracting business rules out of a Java application. To achieve this goal, the initial step is the model discovery from Java code followed by variable classification and domain variables model creation. The final step of their approach is to extract the business rules model from the domain variables model.

An extensive research on application of model-driven development approach to smart contracts in general was produced in [26]. In this research study, the author conducted a literature survey to gain the requirements and techniques to formulate a model-driven development method for smart contracts development. The proposed model-driven method is evaluated using a standard case study and an experiment involving smart contract developers. However, this research study does not support the frequently evolving programming language requirements and established standards to the smart contracts development.
An initial step to applying model-driven engineering to generate smart contracts for IoT and cyber-physical systems by Garamvolgyi et al. talks coordinating the usage of aforementioned system elements from behavioral models, specifically UML statecharts.[6]

A DEMO, BPMN and UML based approach by Skotnica et al. proposes a visual domain-specific language called DasContract. This approach creates models that contain all possible execution paths according to the DEMO transaction axiom and considers modeling of smart contracts using three subsystems - data, processes and forms. [27]

An agent-based approach by Frantz et al. is a computational modeling paradigm in which phenomena are modeled as dynamical systems of interacting agents.[28]

An MDE tool, Lorikeet by Weber et al. [29] uses BPMN models and fungible/non-fungible registry data schema along with the bpmn-js modeling library to develop a business process execution and asset management focused approach for developing smart contracts. Another approach lead by Weber, [30] uses Camunda which is an open-source workflow and decision automation platform to model business processes to be executed using smart contracts.

The behavioral modeling of smart contracts is realised by Mavridou et al. [31], which proposes using finite state machine based approach to model smart contracts execution. This tool is built upon WebGME MDE tool and provides a graphical editor along with automatic code generation with security plugins.

A recent study by Hamdaqa et al. [32] propose a feature-oriented domain analysis technique, iContractML, to extract the a reference model based on the similarities and differences between the three blockchain platforms namely, IBM Hyperledger Composer, Azure Blockchain Workbench, and Ethereum. This research further realizes the derived reference model into an MDE framework to enable smart contract developers in modeling and generating smart contracts that are compatible with multiple blockchain platforms. Our approach differs from iContractML extensively as the technique used to extract domain models in our approach uses reverse engineering of frequently used code patterns, whereas, iContractML uses a forward engineering approach for domain analysis. The motivation behind iContractML is to promote scalability across different blockchain platforms and our approach focuses on realizing the most frequently used code patterns by smart contract developers in the form of a library of models, thereby, enabling secure reuse of code libraries and better comprehension of smart contract functionalities.

6 THREATS TO VALIDITY

Threats to Internal Validity

One of the factors that can affect our results is that our approach is a code clone based domain model mining method which can only derive instances of repetitively used domain models in smart contracts. Though our method can account for variations in models, this cannot be relied on for extending the current state of usability of smart contracts. To overcome this limitation of allowing only a threshold of variation in domain models during the code clone detection method, we analysed and updated the derived domain models depending on their applicability in a specific business process. We recognize that further research in the area of business process modelling for Ethereum smart contracts may benefit our results.

Threats to External Validity

The dataset analyzed in this research consisted of a subset of all the Solidity smart contracts deployed onto the Ethereum main-net that corresponds to DApps. Our dataset does not include the smart contracts that do not correspond to a DApp deployed onto the Ethereum main-net. Moreover, DApps are being built on top of other blockchain networks.

Manuscript submitted to ACM
as well and might represent different characteristics when compared to the smart contracts analyzed in this research. We recognize that further research with a larger dataset may strengthen the correlation established in our results. We highlight that the contribution of this paper is to conduct an initial study towards producing an MDE framework to efficiently develop DApps that are maintainable and secure. However, it should be considered that our research results may not be applicable to all the DApps built on top of other blockchain networks as well.

7 CONCLUSION AND FUTURE WORK

The main reason behind code cloning in Ethereum Smart Contracts is the stigma around the complexity involved behind securing a smart contract and using the features provided by the Solidity programming language and the Ethereum blockchain technology to efficiently develop a decentralised application (DApp). The lack of experience, resources and an active involvement of online community in developing DApps have made the software developers of traditional programming languages to rely on standards and existing deployed DApps available, thereby making the Ethereum smart contracts ecosystem very homogeneous in functionality. We conducted this exploratory analysis of code clones in Ethereum Smart Contracts as an initial step towards developing a model-driven development framework for developing smart contracts based DApps. We perform code clone detection of Type 1, 2, 2c, 3-1, and 3-2c using the NiCad code clone detection tool. We especially leverage the the near-miss code clone detection of Type 3-1, 3-2c provided by NiCad to understand the functionalities implemented by various smart contracts of a similar background or domain to semantically categorise the highly cloned code clusters.

The future work in this research would be to perform an analysis of use-cases implemented by the domains listed in our categorization of code clones, to be factored in the structural domain models of smart contracts through M2M transformations for each of the listed domain categories. Eventually we aim at producing a model-driven development framework for fast and efficient development of DApps with increased maintainability, extensibility and diversity of business use-cases.
REFERENCES

[1] Solidity, "Solidity home page," https://solidity.readthedocs.io/en/v0.5.1/, Accessed: 2021.
[2] J. Chen, M. Alalfi, T. Dean, and Y. Zou, "Detecting android malware using clone detection," Journal of Computer Science and Technology, vol. 30, pp. 942–956, 09 2015.
[3] C. K. Roy, "Detection and analysis of near-miss software clones," in 2009 IEEE International Conference on Software Maintenance, 2009, pp. 447–450.
[4] Ethereum, "Ethereum home page," https://www.ethereum.org/, Accessed: 2021.
[5] E. I. P. (EIP), "Ethereum improvement proposals (eip)," http://eips.ethereum.org/., Accessed: 2021.
[6] P. Garamvölgyi, I. Kocsis, B. Gehl, and A. Klenik, "Towards model-driven engineering of smart contracts for cyber-physical systems," in 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2018, pp. 134–139.
[7] Etherscan, "Etherscan home page," https://etherscan.io/., Accessed: 2021.
[8] S. O. D. H. Page, "State of the dapps home page," https://stateofthedapps.com/., Accessed: 2021.
[9] DAppRadar, "Dappradar home page," https://dappradar.com/., Accessed: 2021.
[10] Sol2Uml, "Sol2uml home page," https://www.npmjs.com/package/sol2uml/v/1.1.6, Accessed: 2021.
[11] OpenZeppelin, "Openzeppelin home page," https://openzeppelin.com/., Accessed: 2021.
[12] CoinBase, "Coinbase home page," http://www.coinbase.com/., Accessed: 2021.
[13] Kraken, "Kraken home page," http://www.kraken.com/., Accessed: 2021.
[14] Cryptowatch, "Cryptowatch home page," http://cryptowatch.ca., Accessed: 2021.
[15] ECDSA, "Elliptical curve digital signature algorithm," Accessed: 2021.
[16] A. Infinity, "Axie whitepaper," https://whitepaper.axieinfinity.com/., Accessed: 2021.
[17] Decentraland, "Decentraland whitepaper," https://docs.decentraland.org/blockchain-integration/create-a-dapp/, Accessed: 2021.
[18] Provable, "Provable home page," http://www.provable.com/., Accessed: 2021.
[19] ChainLink, "Chainlink whitepaper," https://link.smartcontract.com/whitepaper, Accessed: 2021.
[20] TXL, "Txl home page," http://txl.ca/., Accessed: 2021.
[21] M. Kondo, G. Misherghi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," in 29th International Conference on Software Engineering (ICSE 2007), 2007, pp. 96–105.
[22] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," in 9th International Conference on Software Engineering (ICSE 2007), 2007, pp. 96–105.
[23] Z. Gao, X. Xia, V. Jayasundara, D. Lo, L. Jiang, and J. Grundy, "Smartemded: a tool for clone and bug detection in smart contracts through structural code embedding," in Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSM 2019. USA: IEEE, Institute of Electrical and Electronics Engineers, 2019, pp. 394–397.
[24] H. Liu, Z. Yang, Y. Jiang, W. Zhao, and J. Sun, "Enabling clone detection for ethereum via smart contract birthmarks," in Proceedings of the 27th International Conference on Program Comprehension, ser. ICP' 2019. IEEE Press, 2019, p. 105–115.
[25] V. Cosentino, J. Cabot, P. Albert, P. Bauquel, and J. Peronnet, "A model driven reverse engineering framework for extracting business rules out of a java application," 08 2012.
[26] K. Boogaard, "A model-driven approach to smart contract development," in Utrecht University, 2018.
[27] M. Skotnicna, J. Klicpera, and R. Perger, "Towards model-driven smart contract systems - code generation and improving expressivity of smart contract modeling," 03 2021.
[28] C. K. Frantz and M. Nowostawski, "From institutions to code: Towards automated generation of smart contracts," in 2016 IEEE 1st International Workshops on Foundations and Applications of Self Systems (FAS’W), 2016, pp. 210–215.
[29] A. B. Tran, Q. Lu, and I. Weber, "Loriket: A model-driven engineering tool for blockchain-based business process execution and asset management," in Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018 co-located with 15th International Conference on Business Process Management (BPM 2018), Sydney, Australia, September 9-14, 2018, ser. CEUR Workshop Proceedings, vol. 2196. CEUR-WS.org, 2018, pp. 56–60.
[30] O. L’oper-Pintado, Q. Lu, and I. Weber, "Caterpillar: A blockchain-based business process management system," in Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2017 co-located with 16th International Conference on Business Process Management (BPM 2017), Sydney, Australia, 2017, ser. CEUR Workshop Proceedings. CEUR-WS.org, 2017.
[31] A. Marvindou and A. Laszka, "Designing secure ethereum smart contracts: A finite state machine based approach," CoRR, vol. abs/1711.09327, 2017.
[32] M. Hamdaqa, L. A. P. Metz, and I. Qasse, "Icontractml: A domain-specific language for modeling and deploying smart contracts onto multiple blockchain platforms," in Proceedings of the 12th System Analysis and Modelling Conference, ser. SAM 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 34–43.
[33] TXL, "Txl home page," http://txl.ca/., Accessed: 2021.
[34] 0x Protocol, "0x protocol whitepaper," https://github.com/0xProject/whitepaper/blob/master/0xwhitepaper.pdf, Accessed: 2021.
[35] E. N. Space, "Ens whitepaper," https://status.im/whitepaper.pdf, Accessed: 2021.
[36] B. Org, "Beat org whitepaper," , Accessed: 2021.

Manuscript submitted to ACM
[37] M. Jurgelaitis, V. Drunglas, L. Ceponiene, R. Butkiene, and E. Vaiciukynas, "Modelling principles for blockchain-based implementation of business or scientific processes," in IVUS, 2019.

[38] OMG, "Object management group," https://www.omg.org/, Accessed: 2021.

[39] K. Boogaard, "A model-driven approach to smart contract development," 2018.

[40] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, "Untrusted business process monitoring and execution using blockchain," 09 2016.

[41] H. Syahputra and H. Weigand, The Development of Smart Contracts for Heterogeneous Blockchains, 01 2019, pp. 229–238.

[42] C. K. Frantz and M. Nowostawski, "From institutions to code: Towards automated generation of smart contracts," in 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W), 2016, pp. 210–215.