A Tool-Supported Approach to Smart Contract Creation, Validation and Analysis

Master Thesis

Supervisor: Michael Felderer

Raphael Gwiggner (1215661)

raphael.gwiggner@gmail.com

Innsbruck
7 October 2019
Abstract

Since the beginning of Bitcoin in 2009 many other cryptocurrencies have followed. Not only businesses and communities are interested in these new currencies, but also private users buy, sell and trade with them. One of them is Ethereum. Ethereum owns a big advantage over Bitcoin: it offers smart contracts that include turing complete scripting languages such as Solidity and Serpent. With smart contracts, no third parties between a buyer and a seller are needed. In this paper common mistakes in writing smart contracts have been explored, how to fix them and show an overview of financial frauds like Ponzi Schemes. Also, a self-developed tool called Smartract CVA will be presented. This tool is able to create, validate and analyze smart contracts and helps users to get an assessment about Initial Coin Offerings. For our application different steps like completing a questionnaire, reading of a whitepaper and reproducing a smart contract with the help of a self-created Domain-Specific Language (DSL) are required.
Contents

1. Introduction ........................................... 4
   1.1. Problem Statement .................................. 5
   1.2. Research Approach ................................ 5
   1.3. Research Questions ................................ 6

2. Related Work ........................................... 7

3. Background and Overview of Cryptocurrencies/Smart Contracts ........ 8
   3.1. Dezentralized Systems and Ethereum .................. 8
   3.2. Background of Smart Contracts ........................ 8
   3.3. Example: CryptoKitties ............................... 10
   3.4. Example: Bancor ..................................... 11
      3.5.1. Tron ........................................... 13
   3.5. Ethereum’s Smart Contract Competitors .............. 13
      3.5.2. EOS ............................................ 14
   3.6. Initial Coin Offerings ............................... 15
      3.6.1. What are Initial Coin Offerings? ................. 15
      3.6.2. Evaluate Initial Coin Offerings ................ 15
      3.6.3. Known Pitfalls ................................ 17
      3.6.4. Ponzi Schemes ................................ 22
      3.6.5. Ponzi Identifikation ............................ 23
      3.6.6. Ponzi Analysis ................................ 23
      3.7.1. Structure of a simple Smart Contract ............ 26
      3.7.2. Smart Contract of an ICO ........................ 27
      3.6.7. Ponzi Statistics ................................ 26
      3.7. A Smart Contract in Detail ......................... 26

4. Smartract CVA: a Smart Contract Creation, Validation and Analysing Tool 31
   4.1. Design and Architecture ............................. 31
      4.1.1. Environment .................................... 31
   4.2. Smartract DSL ....................................... 32
   4.3. Steps of Contract Validator .......................... 33
      4.3.1. Whitepaper .................................... 34
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Genetic Algorithm</td>
<td>11</td>
</tr>
<tr>
<td>3.3</td>
<td>Dapp statistics of 2018 from dapp.com</td>
<td>14</td>
</tr>
<tr>
<td>3.4</td>
<td>Top ten ponzi schemes</td>
<td>26</td>
</tr>
<tr>
<td>3.5</td>
<td>Voting diagram of adChain Registry</td>
<td>28</td>
</tr>
<tr>
<td>4.1</td>
<td>Smartract API by etherscan.io</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>Transaction results</td>
<td>40</td>
</tr>
<tr>
<td>4.3</td>
<td>Grammar rules of Smartract DSL</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>TAM</td>
<td>58</td>
</tr>
</tbody>
</table>
Listings

3.1. simple call to itself or another contract ............................................. 17
3.2. player_input function ........................................................................ 18
3.3. Solution for the player_function problem .................................................. 18
3.4. Simplified version of "King of the Ether Throne" ....................................... 19
3.5. Participants donate Ether to sponsor contracts at their choice ................. 20
3.6. This attack can steal all the Ether from the SimpleDAO ....................... 21
3.7. Modified version of contract Mallory ...................................................... 21
3.8. Example of a handover scheme ............................................................... 24
3.10. Sample how a ICO contract could look like ........................................... 26
3.11. Excerpt of the smart contract sale.sol ................................................... 29

4.1. Sample for an Input DSL ....................................................................... 35
4.2. Generated smart contract with the input provided in listing Listing 4.1 ....... 37
4.3. Example of a handover scheme ............................................................... 42
4.4. Initialization of jinja2 .............................................................................. 42
4.5. Different operators .................................................................................. 42
4.6. Jinja Filter .............................................................................................. 43
4.7. Source output folder .............................................................................. 43
4.8. Create output folder .............................................................................. 43
4.9. Create output folder .............................................................................. 43
4.10. Create model from file ......................................................................... 43
4.11. Write solidity file .................................................................................. 44
4.12. Declaration of a contract ...................................................................... 44
4.13. Print global variables .......................................................................... 44
4.14. Print all events and their params .......................................................... 44
4.16. ContractModel ..................................................................................... 46
4.17. Contract rules ....................................................................................... 46
4.18. Modifier rule ........................................................................................ 47
4.19. Function block rules ............................................................................ 47
4.20. Assignment rules ................................................................................ 48
4.21. If-condition rule .................................................................................. 48
4.22. Declaration of a contract ..................................................................... 48
4.23. For-loop rules .......................................................... 48
4.24. Small DSL example .................................................. 49
1. Introduction

In 2009 the first decentralized cryptocurrency called Bitcoin has been created. Many other blockchain based cryptocurrencies as Ethereum and Litecoin followed. The rise of Bitcoin and other alternative cryptocurrencies called altcoins is remarkable and received public attention. Altcoins like Ethereum extend the basic design of Bitcoin by offering so-called ”smart contracts”. Smart contracts are contracts between buyers and sellers without any third parties. A programming language like Solidity is used for writing smart contracts that specify rules for transactions. These transactions can be tracked and are irreversible.

The aim of this paper is to understand what smart contracts are, how they behave, their security issues and how to fix them. Also, a tool called Smartract CVA has been created according to this master thesis. In chapter 2 we take a look at related work and how they influenced this paper. Chapter 3 deals with a short overview of decentralized systems and smart contracts. Initial Coin Offerings will be explained in detail. In connection with ICOs, financial frauds like Ponzi schemes can be made by cheaters. This system has made its way to cryptocurrencies especially with the functionality of smart contracts. To avoid such fraud systems a tool called Smartract CVA has been developed. Smartract CVA will be explained in chapter 4. All functions this tool offers will be discussed.

Smartract CVA is a small web application where you have the opportunity to create, validate and analyze smart contracts. We discuss the design and architecture of the web application. To generate a smart contract a DSL is needed. This DSL has been created with TextX, which is also a DSL for Python. In this paper, we also call our own created DSL Smartract DSL to avoid any confusion.

Smartract DSL is able to handle all different input data types like Integer, Float, String, Identifiers and also Objects. This DSL supports components like ”if conditions”, ”functions” and ”for loops”. With the help of the described tool, investors get support about deciding over an ICO, if it is only a scam or maybe something valuable. In the next chapter, we evaluate our results of this masters thesis and give a brief reflection on our research questions. Also, some failed approached and workarounds will be mentioned. Last, as in every thesis, a summary with all highlights and future work is provided. We debate about how to improve our Smartract CVA and make the Smartract DSL more precise.
1.1. Problem Statement

The key parts of the cryptocurrency markets are Initial Coin Offerings also called ICOs. According to CNBC\(^1\) this type of crowdfunding has raised over 12$ billion in 2018 till August. The North American Securities Administrators Association (NASAA) has started an operation called *Operation Cryptosweep*. This operation should identify scam projects so investors do not throw away their money. In May 2018 over 70 cases have been detected. In August 2017 the number has been expanded to 200.

However, it is not easy to detect cheating systems. According to Investopedia\(^2\) nearly 80 percent of Initial Coin Offerings are scams. In response to this problem, our work investigates several options to identify a fraud project and give investors some useful hints. We give some examples how a cheating smart contract could look like and also consider different patterns of fiddles. *Smartract CVA*, our own developed tool, should help investors to make a capital expenditure decision and detect such fraud systems, which is the aim of the proposed research.

1.2. Research Approach

For this paper, we divide our research approach into two parts.

The first one deals with the theoretical part. All began with examining different papers over various scientific platforms (from Google’s Scholar\(^3\) till Base\(^4\)) referring to cryptocurrency. We picked the best one, that had a tight reference to any of the following keywords:

- Smart Contract
- Initial Coin Offering
- Ponzi Schemes (in relation to cryptocurrency)
- Ethereum (only for background information)
- Crowdfunding (in combination with the Blockchain or Ethereum)

After finding some useful papers a closer look at ICOs and their fraud systems has been taken. With the paper of Massimo Bartoletti et al.[4] a huge material covering this topic was given. Different types of cheating systems have been explained and are also a part of this work.

---


\(^3\)https://scholar.google.at/

\(^4\)http://de.base-search.net/
The other part cares about the coding part. For this, we had to think about how we can help to identify mock projects. We came up with the idea of developing a small tool, that should help to review a blockchain based crowdfunding project. Various questions should be answered and a contract of a sample how a smart contract of the ICO could look like will be generated. This can be compared with the original one of the project. If no original smart contract can be found, a look at etherscan.io can help to find the EVM code. With the help of the tool Porosity [18] blockchain-based smart contracts bytecode can be decompiled to smart contracts.

1.3. Research Questions

This paper deals with important research questions that influence today's cryptocurrency market. Following research questions will be examined in this thesis:

- What are common used fraud systems referring to smart contracts and how do they affect reputation and crowdsales of ICOs?

- Is there a possibility to really guarantee that an ICO is doing what they promise? If yes, how can we identify and prevent them?

- Are we able to develop a tool that helps users and investors avoiding scam ICOs?
2. Related Work

Due to the noeventy of Ethereum (2015) there has not been much work on the analysis of smart contracts. Since the rise of Bitcoin, Ethereum and other altcoins, more and more researchers have set their focus on the growth of cryptocurrency and the blockchain technology. With the possibilities of smart contract also security issues have been explored. With the DAO-Attack (see chapter 3.6.3) users tried to find ways to manipulate smart contracts. References to the DAO-Attack can be found at Coindesk [2] and medium.com [12]. The paper from Kevin Delmolino et al. [9] gives a short insight about creating safe smart contracts step by step.

Several recent papers have focused on security issues referring to smart contracts. M. Bartoletti et al. show with their publications that this is an unexplored topic and show some new research on it. Our approach follows on the paper An Empirical Analysis of Smart Contracts: Platforms, Applications and Design Patterns [5] which gave the base how to analyse smart contracts in detail.

Other useful papers from M. Bartoletti et al. are A Survey of Attacks on Ethereum Smart Contracts [3] and Dissecting Ponzi Schemes on Ethereum: Identification, Analysis and Impact. Both papers gave a lot of information about different schemes of fraud systems. The second mentioned one describes the methodology for identifying Ponzi schemes. First, they collect information on associated websites like etherchain.org/contracts and etherscan.io. After this they inspected different sources like Reddit, Google and official blogs from the providers. They seeked for keywords like "Ponzi", "pyramid", "scam", and many more.

Also a small study has been made and shows how less ICOs pay their investors money back (with only a small amount of profit). Maughelli Francesco [17] made a small publication according to this topic with security related experiences of smart contracts over the Ethereum blockchain.

Many ICOs do not share their smart contracts written code. Only the compiled code (EVM bytecode in this case) is available. To get the original smart contract a tool called Porosity [18] from comae.io has been developed. This tool will also be used to decompile EVM code back into its Solidity code.
3. Background and Overview of Cryptocurrencies/Smart Contracts

This section gives a brief overview of decentralized systems in general, Ethereum, design and examples of smart contracts and their pitfalls.

3.1. Dezentralized Systems and Ethereum

One important concept of cryptocurrencies is decentralization. A decentralized system is the base of most cryptocurrency where users can transfer money and interact with contracts. To understand how it works it is useful to understand the connection between decentralization and Bitcoin et al. A decentralized system requires multiple participants (peers) to make their own independent decisions. On the other hand a centralized system has a single authorized node that makes the decision on behalf of all participants (peers). Peers are able to interact directly with each other or share information without a single centralized authority.

An example for decentralization is an email. An email relies on its core as a decentralized system based on the Simple Mail Transfer Protocol (SMTP). But also this open standard is not purely decentralized. Anyone can operate an email server on their own. A few companies control most of the mail providers.

Figure 3.8 shows a high-level structure of a decentralized system. Nodes (peers) build a network, that stores required data, handles needed information and updates data by publishing transactions [9]. The main part of a cryptocurrency’s decentralized system is called blockchain. The blockchain stores the transactions in a partial or total ordering agreed by users. The whole data can be copied by all users if it is public and permissions are granted. The decentralized system tracks the balance of each user by associating each unit of currency to an address. The address represents a hash of a public key and can be seen as a wallet. To administrate the "wallet", a private key is needed, so a user is able to spend money correlated to that address.

3.2. Background of Smart Contracts

There exists a big difference compared to the Bitcoin technology. The scripting language of Bitcoin is limited by its expressiveness. In other words, Bitcoin is using a not turing complete
3.2 Background of Smart Contracts

Figure 3.1.: High-level structure: Users are able to receive and send data to a contract and vice versa. Details of smart contracts are stored in the public blockchain. The execution and update of a smart contract program will be handled by numerous miners who reach consensus over the result. [9]
language called *Script* [14]. On contrary Ethereum offers the possibility to program smart contracts with turing complete languages like Serpent or Solidity [7], which bypass those limits. After deploying a smart contract it is permanent, so there is no way to alter the code except by destroying it. A storage file, program code and the account balance are the main parts of such a contract. Smart contracts use the same technology as shown in figure 3.8. All important information are stored on the public blockchain (storage file). The execution takes part by various miners who update the blockchain afterwards. The contract invokes with a transaction which can be from a user or another contract (defined in its program code). This can be seen as a function call. Contracts are able to read/write to their existing storage files or send and receive money to other contracts/users.

To avoid endless loops, pay fees and encourage efficiency in the protocol code Ethereum uses a concept called *gas*. Gas can be seen as a kind of contract fees. As mentioned before every transaction must be executed by the network of miners. They have to validate and execute each contracts code correctly and update accordingly. At this point contracts can contain infinite loops (cause denial-of-service attacks) or require expensive storage or computations [5]. Hence every single operation demands some gas. The amount of gas depends on the complexity of an instruction. Simple mathematical operations like additions are cheaper than writing to the storage file. For example a single calculation needs a small amount of gas, on the other hand thousands of instructions require more gas. If the transaction doesn’t contain enough gas to execute all instructions, it stops immediately and the applied gas is kept by the miners [3].

Smart contracts offer the opportunity to send from one contract to another. So multiple contracts are able to interact with each other. Ethereum limits the call-stack size to fixed 1024 calls. If this limit is reached, no other contract can be appended.

3.3. Example: CryptoKitties

From financial contracts over insurance policies to multiplayer online games - smart contracts provide a huge range of possible applications. One of the most popular games using the smart contract technology is called CryptoKitties [1]. The first test version of CryptoKitties took place on October 19th 2017. It is one of the first games using the blockchain technology. Cryptokitties is a game were users collect, buy, sell, trade or breed virtual, adorable cats. Every single cat is unique and one-of-a-kind. According to their website over 180.000 users are registered and have spent over $20M in Ether. These so called *kitties* are created by code and bred by paying Ether on smart contacts to generate a new cat.
3.4 Example: Bancor

How does it work?

As mentioned before CryptoKitties is powered by the Ethereum blockchain. Ethereum offers the possibility to develop Decentralized Applications (DApp), thus complex logic can be used within a transaction. Decentralized Applications do not require a middleman such as the Google Play or Apple App Store. They connect users and providers directly. Further, there exists a huge potential to create transactions far beyond only sending money from one user to another. For instance casino games like poker or roulette.

CryptoKitties are based on the Genetic Algorithm [22]. So different attributes of a cat like color, type, fur, parents, children and facial expression can be generated/stored. It’s comparable with the real DNA of humans. The Genetic Algorithm behaves the same, except the DNA is represented in a binary code. One half of the father’s genome and the other half of the mother’s genome will be copied into a new, unique child genome. This process is called crossover 3.2.

With the help of smart contracts only the binary DNA of the parents is needed. The calculation itself is a simple swap. The CryptoKitty contract has something common with many ICOs (initial coin offerings). Users have to invest and receive cats instead of tokens. Around 5k "promo" cats are given away for free so the starting community owns something valuable and rare further along the line. More about ICOs will be discussed in section 3.6

3.4. Example: Bancor

Another example of interacting with smart contracts is Bancor\(^1\). Bancor is a decentralized app over the Ethereum blockchain. It allows users to convert between two tokens without any third party. Over hundred different tokens are offered. Bancor provides so called smart tokens. A smart token contains a balance of another reserve token like Ether. Smart tokens will be created when purchased and destroyed when cashed out. Therefore a smart token will always be purchased with its reserve token, where the reserve token holds the current price.

\(^1\)https://about.bancor.network/
Bancor uses following formular to determine the current price:

\[
\text{price} = \frac{\text{balance}}{\text{supply}} \times \text{crr}
\]

- **balance**: total ether which is locked to the contract
- **supply**: tokens locked into the contract
- **crr (constant reserve ratio)**: this value is set by the smart token creator.

To demonstrate the formula in practice assume a new, fictitious project. This project contains a game with a token crowdsale of $1,000,000 in Gametokens (GAT). To simplify the example 1 GAT equals 1 Ether. The current price of Ether is $100 and the game costs $55. Now a user pays $40 to the project that equals a balance of 0.40 GAT. After the game releases or the crowdfunding ends the user is able to buy the game or however he does not want to own a copy of the game he can cash out the 0.40 GAT to 0.40 ETH with the help of Bancors smart token contracts.

The crowdsale ends and has been a success with $1,100,000 equals to 11,000 GAT. As mentioned before the project creator issues the 0.40 GAT to each investor. 22,000 GAT will be created and 11,000 GAT are going to the investors. The rest are reserved for the Bancor contract. 90% of the raised fund will be used for the video games, the other 10% are used to collateralize the Bancor contract. All facts stated again:

- **$1,000,000** = target, that had been reached in the crowdsale
- **GAT 11,000** = issued to users
- **GAT 11,000** = content of the contract - can be seen as supply
- **$900,000** = financial calculation for development
- **ETH 1,000** = remaining as balance into the Bancor contract

So we receive the following formula:

\[
1GC = 1ETH = \frac{1100ETH}{11000GAT} \times 10\%
\]

As mentioned before people can also sell their GAT credits. 500 users are not interested in the game anymore so they are going to sell their 0.4 GAT. The new formula would be:

\[
1GC = 1ETH = \frac{(1100ETH \times 200ETH)}{(11000GAT \times 200GAT)} \times 10\%
\]
So the value of 1 GC falls lightly but the users almost get back their previous investment. Of course this scenario can be shown reverse. Around 2000 new people are interested in this cool, new game and they can still buy GAT the formula would look as following:

\[
1GC = 1ETH = \frac{900ETH + 800ETH}{10800GAT + 800GAT} \times 10\%
\]

\[
1GC = 1.47ETH = \frac{1700ETH}{11600GAT} \times 10\%
\]

3.5. Ethereum’s Smart Contract Competitors

More and more blockchains are based on the smart contract platform. Still in 2019 Ethereum is the most valuable smart contract technology, but more and more competitors are on their way. In this section a short overview of two other smart contract platforms is given.

3.5.1. Tron

Tron (TRX) is a decentralized app platform that can be best compared to Ethereum. In 2018 ETH had the most total number of Dapps and EOS reached the highest volume. Tron had no chance against the big player. According to dapp.review Tron is now the leading platform regarding volume\(^2\) and active users in June 2019.

Tron has many benefits over Ethereum. One advantage can be seen as the programming language. Smart Contracts in Ethereum require newer languages like Solidity. Tron is permanently developed with the Java language, which is one of the most wide-spreaded programming languages.

Ethereum is able to handle approximately 15 transactions per second. According to Justin Sun, the founder of Tron, Tron is able to handle 100,000 transactions per second\(^3\). Another advantage are the fees. Ethereum fees are high, whereas Tron does not charge fees. Tron relies on the Delegated Proof of Stake (DPoS) protocol. Only a limited amount of privileged users are able to create blocks. Generally those users are elected by the community in a voting process. DPoS is fast, cheap and more reliable.

As every other infrastructure also Tron’s Community must be paid. Tron holders have the opportunity to freeze TRX to gain power/energy and bandwidth. Power can be represented

---

\(^2\)https://analytics.dapp.review/
\(^3\)https://www.abra.com/blog/crypto-bites-a-chat-with-ethereum-founder-vitalik-buterin/
as your vote for the super representatives (SRs), which are responsible for the block production. Bandwidth points are used to fulfill transactions. Every transaction on the tron network requires bandwidth points. The Tron network provides 5000 free bandwidth points for each account every 24 hours. This are around 25 free transactions per day. If more transactions are required the user needs to obtain more bandwidth points. This can be achieved by freezing TRX or pay a fee for the transaction. Freezing a Tron token will lock the token for a certain number of days, which means the trader is not able to sell or trade the tokens until they are unfrozen again. In exchange the user will be rewarded with bandwidth or energy points.

![Dapp Status of Most Active Blockchains 2018](image)

3.5.2. EOS

In 2017 a new blockchain platform has been created, called EOS. The biggest advantage over Ethereum is again the scalability. The EOS blockchain can handle millions of transactions per second. Also a Delegated Proof of Stake concept is used. EOS also has no transaction fees but instead uses inflation. The annual inflation rate is 5%. Token holders can vote on increasing or decreasing the annual inflation rate. The inflation itself is split into two groups: the block producers and the worker proposals. Both will be rewarded every time a block is produced. EOS platform requires C++ as programming language. As mentioned in the subsection before Tron requires newer languages.
3.6. Initial Coin Offerings

This section starts with a small introduction about Initial Coin Offerings (ICOs). It counts some of the most famous ICOs and discusses the juristic side in some nations. There are a lot of risks and fraudless systems concerning to ICOs (like Ponzi schemes 3.6.4) that you can see in section 3.6.6.

3.6.1. What are Initial Coin Offerings?

Initial Coin Offerings (ICOs) can be compared to Initial Public Offerings (IPOs). All ICOs are based on crowdfunding with cryptocurrency. A part of the collected cryptocurrency is hold for the investors in form of so called tokens. When the funding goal of an ICO is achieved, these tokens can get a functional behavior within the project of the ICOs. Thus, it is a great opportunity for new startups to raise capital quicker and easier. There is no unnecessary paperwork required. Nethertheless the ICO should offer a so called ”white paper”, which contains all the details about the project. According to coindesk.com 4 over $3,775M have been funded to ICOs with an average monthly new ICO funding around $500M. In 2013 the first ICO token sale was held by Mastercoin, a digital currency and communication protocol built on the blockchain, and reached over $2.3M. The real hype began in 2017. Over 230 ICOs were tracked at the end of the year referring to coindesk.com.

ICOs are risky and speculative investments. Although little paperwork is needed also a lot of scammers are attracted. They make their project look more appealing in the white paper than it is actually possible to implement. With the investment in an ICO you also invest in the idea of a project. So basically you can not be sure if their intention will succeed.

At first sight it is not always clear if the ICO is serious or a scam and often offers no protections for investors. China for example has officially banned seven ICOs and insisted that all investors should get their investments back [6]. Canada is working on regulating ICOs. South Korea prohibited ICOs in September 2017. The Arab Emirates and other countries issued official guidance on ICOs.

3.6.2. Evaluate Initial Coin Offerings

As explained in the previous section an initial coin offering is a ”fundraising mechanism” in which the new project sells own created crypto tokens. Not all ICOs have a good will in common. As a part of this work, a tool for identifying bad ICOs has been developed which will be the focus in the next chapter. The tool itself does not guarantee a fraudless system. People who want to invest in an ICO should create a framework to evaluate the ICO. Either similar

---

4https://www.coindesk.com/ico-tracker/
methods like described in 3.6.5 are used or the following steps serve non experienced users as a simple start:

**Step 1 - White paper**

First, take a look at the white paper. The white paper of an ICO can be found on the official website of the project, at a github repository or at coinlauncher.io, a white paper database, that contains thousands of projects. If the project does not have a white paper it may be untrustworthy. Typically investors do not read through the whole white paper. It is not necessary to read everything. A look at the main parts should be enough. After going through the paper you should be able to answer if the project conforms with your preconceptions. Some ICOs promise impossible implementations, such as developing a new social network which will be bigger than Facebook, a new Google and many more. Check if they already launched a product. If this is the case, observe the current stage of the project.

What if the ICO does not reach the soft cap? The soft cap is the minimum financial goal they set for the crowd sale. Some companies do not refund the money of investors back. Check, if you are able to get your investment back using a push/pull mechanism (e.g. Open-zeppelin).

**Step 2 - Team & Collaborations**

Look up each team member with all social media networks you know. Google them, take a look at career platforms like LinkedIn or Xing. If it is a member of the the development team, Github is a great bet. Find out if they share any experience with crypto assets, Ethereum, smart contracts or the blockchain in whole. Another important thing is to review if there is no management or developer with a dark past (i.e. Ponzi systems or other rip-off projects). Red flag an ICO with more advisors than team members.

**Step 3 - Community**

Sources like Facebook, Medium, Twitter, Reddit or bitcoinTalk.org can help evaluating a project. Verify if users who spread positive news about the ICO are really trustworthy and not only fake. Is the project team an active part of the community? Do they share updates with them and answer questions rapidly? Do not let a fool be made of you, if the ICO is changing their terms over and over again.

**Step 4 - Code**

Assuming you have some programming skills, check the smart contract for your own. How is the quality of code? Do they even offer open-source code? If you are not familiar with coding.

5https://coinlauncher.io/white-papers
Check if there are some comments or use a tool like contract validator.

**Step 5 - Tokens**

Tokens add value to the platform if they are short. Too many can be a sign that the project leaders only want to raise money with tokens. Next check the token network effect. Do they have any relevant events? Tokens should be a main and important part of the venture. Why do they need tokens? Can not the project be fulfilled with Ethereum or Bitcoin? Watch out for the token distribution. Do they have minable tokens or only set to the standard ERC20 token?

### 3.6.3. Known Pitfalls

"With great power comes great responsibility" once Benjamin Parker said to his nephew. Smart contracts are a powerful blockchain technology with many opportunities, but also present some hazards when programmed wrong. This section highlights some of the typical pitfalls when programming smart contracts.

#### Vulnerabilities in general

N. Atzei et al. [3] classifies the vulnerabilities of Ethereum’s smart contracts into three groups, according to their level (Solidity, EVM bytecode and the blockchain). Different vulnerabilities are illustrated through listings. The detected weaknesses lead to attacks like stealing tokens. When a user invokes a contract without providing a function name or any relevant data a *callback function* is called. This callback functions can be used for criminal thoughts. Listing 3.1 shows an example how Ether is transferred to the callee. *Amount* determines how many Ether/Wei have to be transferred to the receiver c. If the called function does not exist, a callback function of c will be executed. Then the callback function can transfer all submitted Ether to the contract owners address, even if it is not the meaning of this contract.

```solidity
1 c.call.value(amount)(bytes4(sha3("ping(uint256)")),n);
```

Listing 3.1: simple call to itself or another contract

Thus, smart contract code is **immutable** after deployment. Also bugs can no longer be altered, if they are not in a dynamic library. Hence, if a contract contains a bug, there exists no direct way to fix it. This immutability provides various attacks like stealing some Ether. For example if the code transfers tokens to a wrong address.

#### Pitfalls by Example

The University of Maryland taught their under graduated students to program smart contracts and documented their experience. The task was to develop a smart contract application of
their choice, using Serpent as programming language. The students were divided into different
groups and one group decided to create a simple "Rock, Paper, Scissors" game on which the paper [9] took a focus to determine common pitfalls. The game itself contains only two main simple functions:

- **player_input**: The player has to submit with the contract, make a game choice (rock, paper, scissors) and place some money to play

- **finalize**: Contains the logic of the game and returns the winner

One detected mistake leads back to a logical error that leaks money in corner cases. Listing 3.2 shows that only two players are allowed, so if a third player joins and sends money to the contract the money disappears (Line 9). There is no chance to recover it. The condition also says, that the amount of money must be exactly 1000 Wei (denomination of Ether). If not, the code also jumps to Line 9 and the money is gone. Third, one player can wait till the other one has made a choice. Due to the choices are send plaintext, the waiting player can see it before deciding on his movement.

The second problem seems obvious, but the first one can be a real problem. Multiple parties can take part in this game and it is up to the miner to decide the order of the transactions. Listing 3.3 presents the solution of this problem. If there are more than two players or the amount is wrong, the money will be refunded.

```python
1 def player_input(choice):
2     if num_players < 2 and msg.value == 1000:
3         reward += 1000
4         player[num_players].address = msg.sender
5         player[num_players].choice = choice
6         num_players = num_players + 1
7         return(0)
8     else:
9         return(-1)
```

Listing 3.2: player_input function

Next K. Delmolino et al. focus on the problem mentioned before: plaintext input. Transactions are public and can be viewed over the whole network. So cheating when the input of a smart contract (or in this case a game) is a clear text, is not tricky at all. Encryption is the solution for such vulnerabilities. This can be done with a secure hash algorithm.

```python
1 def player_input(commitment):
2     if num_players < 2 and msg.value >= 1000:
3         reward += msg.value
4         player[num_players].address = msg.sender
5         player[num_players].choice = commitment
```

18
num_players = num_players + 1
if msg.value - 1000 > 0:
    send(msg.sender, msg.value - 1000)
    return(0)
else:
    send(msg.sender, msg.value)
    return(-1)

Listing 3.3: Solution for the player_function problem

Another example is the game "King of the Ether Throne". To claim the Ether throne users have to pay an amount of Ether. Suppose for example the current price is 1 Ether. If you want to be in the "Hall of Monarchs" (in other words - listed on their website) you have to pay the current price. After this, the price goes up by 33%. Your reign limit is 14 days. If no one takes over your throne within these days you die and you are not making any profit. The current price will be set to the starting price. Else, if someone wants to throw you from the throne, he has to pay the new price and you will receive that payment. Listing 3.4 displays an excerpt of the contract which was written in Solidity.

```solidity
contract KotEt {
    address public king;
    uint public claimPrice = 100;
    address owner;

    function KotET() {
        owner = msg.sender;
        king = msg.sender;
    }

    function sweepCommission(uint n) {
        owner.send(n);
    }

    function() {
        if(msg.value < claimPrice)
            throw;
        uint compensation =
            calculateCompensation();
        king.send(compensation);
        king = msg.sender;
        claimPrice = calculateNewPrice();
    }

https://www.kingoftheether.com/thrones/kingoftheether/index.html
```
Listing 3.4: Simplified version of "King of the Ether Throne"

When a user sends some Ether to the contract with \texttt{msg.value}, he also starts the execution of the fallback function (Line 15). If the sent amount is not enough, an exception will be thrown. If the value is higher than the current price, a new king will be born and a compensation will be sent to the old king. At first sight everything seems legit but it is not. Line 21 may results in stealing Ether due to not checking the return code of \texttt{send}. As previously mentioned every transaction needs gas and the king’s address can be an expensive fallback where the submitted gas is not enough. Assume that the callback function calls another contract, which calls another one, till the contract run out of gas. If this case becomes true, the compensation belongs to the contract. The solution for this problem can be a \texttt{call} instead of \texttt{send} and by handling it with an exception.

\textbf{The DAO Attack}

DAO, a digital decentralized autonomous organization, was crowdfunded (with over 10.7M ETH) via a token sale in May 2016 \cite{2}. The code of DAO is open source and instantiated on the blockchain (Ethereum). DAO can be compared with an investment fund. Users could vote for different kind of projects with their received DAO tokens. On June 18th 2016 an attacker could heist around $60M. After this a hard-fork of the blockchain was made to make the stolen Ether useless. A simplified version of DAO presents the vulnerabilities of the original one and will be discussed with two attacks which are able to exploit them \cite{3}.

```
1 contract SimpleDAO {
2   mapping (address => unit) public credit;
3
4   function donate(address to) {
5     credit[to] += msg.value;
6   }
7
8   function withdraw(uint amount) {
9     if(credit[msg.sender]>= amount) {
10        msg.sender.call.value(amount)();
11        credit[msg.sender]-=amount;
12      }
14   }
15 }
```

Listing 3.5: Participants donate Ether to sponsor contracts at their choice
The first thing to do for **attack number one** is to publish the contract Mallory(3.6. This contract/attack could snitch all Ether. The criminal user spends some Wei to Mallory and invokes Mallory’s fallback which in turn invokes withdraw. This function transfers the Wei to Mallory. In Line 11 of Listing 3.5 the call function invokes Mallory’s fallback again, thus withdraw will be executed again. As you can see, this ends in a loop and the DAO sends credit to Mallory till some of the following exceptions stop the loop:

- Every transaction needs gas. So there could be too less gas.
- Ethereum sets a limit to the stack size thus the stack can be full.
- The balance of DAO approximates to zero.

```
contract Mallory {
    SimpleDAO public dao = SimpleDAO(0x354...);
    address owner;
    function Mallory(){ owner = msg.sender; }
    function () { dao.withdraw(dao.queryCredit(this));}
    function getJackpot(){ owner.send(this.balance); }
}
```

Listing 3.6: This attack can steal all the Ether from the SimpleDAO

The **second attack** has the same task as attack number one. Only two calls to the callback function are needed. The contract for Mallory has to be modified (see Listing 3.7). The contract has to be provided with a small amount like 1 Wei. After this the foe calls the attack function to donate the Wei to himself and withdraws it. Again call invokes the fallback function of Mallory2 and withdraw afterwards. The second time the fallback does nothing due to the variable `performAttack` is set to false. The credit of Mallory2 is updated twice. In the end `getJackpot` is called, which is able to steal all the Ether from DAO and transfer it to the thief.

```
contract Mallory2 {
    SimpleDAO public dao = SimpleDAO(0x42);
    address owner;
    bool performAttack=true;
    function Mallory2(){
        owner=msg.sender; }
    function attack() {
        dao.donate.value(1)(this);
        dao.withdraw(1); }
    function() {
        if(performAttack) {
            performAttack = false;
            dao.withdraw(1);
        }
    }
}
```

21
function getJackpot()
{
    dao.withdraw(dao.balance);
    owner.send(this.balance);
}

Listing 3.7: Modified version of contract Mallory

The trick behind the code is the condition in Line 12 (3.5). The amount input is sent to the DAO contract before decreasing the credit. Due to both attacks were possible. To get a better understandable and detailed example please take a look at [12].

3.6.4. Ponzi Schemes

The rise of the electronic cryptocurrencies has not only captured the public attention, also criminals showed their interest in finding new ways to transfer money anonymous without being tracked. A fraudulent investment operation called Ponzi scheme (also Ponzi game) has reached the cryptocurrency world. It is an investment program, where users enter a scheme by investing an amount of money. How and if they make profit anyway depends on different rules of the scheme but the basic idea of Ponzi remains the same. Every member must find new participants for the scheme. The U.S. Securities and Exchange Commission explain a Ponzi scheme as follows:

A Ponzi scheme is an investment fraud that involves the payment of purported returns to existing investors from funds contributed by new investors. Ponzi scheme organizers often solicit new investors by promising to invest funds in opportunities claimed to generate high returns with little or no risk. With little or no legitimate earnings, Ponzi schemes require a constant flow of money from new investors to continue. Ponzi schemes inevitably collapse, most often when it becomes difficult to recruit new investors or when a large number of investors ask for their funds to be returned.

Most of Ponzi schemes have something common with a pyramid-shaped topology of users. At level x + 1 the users pay the "entry-fee" or compensation to users at level x. When no new investors are found the system will collapse users at the bottom of these layers will lose their investment. According to a study of [20], Ponzi schemes have made around $7M USD in the period from September 2013 to September 2014. With the initial release of Ethereum in 2015 and the boom of cryptocurrencies in 2017, also the impact of Ponzi schemes has increased proportionally. A Ponzi scheme in smart contracts provides criminals interesting features:

7Source: https://www.sec.gov/spotlight/enf-actions-ponzi.shtml
• The creator of the Ponzi scheme smart contract could stay anonymous to some degree.

• Smart contracts are immutable, so nobody can change or terminate the execution of that scheme. Also, a reversion to refund the victims is not possible (exception: a hard fork).

• Most Investors do not inform carefully before investing in the contract and have no idea that a unfair Ponzi scheme system is behind this contract.

3.6.5. Ponzi Identifikation

To identify Ponzi schemes a methodology is needed. In this section we take a closer look at the procedure of [3]. First Ethereum contracts are needed. They can be downloaded directly via a client like geth or retrieved from public explorer websites. Contracts from etherchain.org which are associated with a name and contracts from etherscan.io with a verified source code are taken. The collection altogether contains 1470 contracts. After this each contract must be inspected manually. Following identifications were made:

• manually inspect the code including comments
• if available, project web pages
• targeted queries on search engines (like Google)
• search for official blogs, Reddit pages or something of that kind and look for keywords like e.g. "Ponzi", "Pyramid", "scam", "fraud"

After this inspection 137 contracts were left, that are available online at goo.gl/CvdxBp

The manual identification does not give a guarantee that all contracts, including a Ponzi scheme were found. The next step is to search for hidden Ponzi schemes. One way to find these schemes is to compare the bytecode of known Ponzi schemes with the Ethereum blockchain. For this, the normalized Levenshtein (NLD) distance measure is used to find the similarity between two bytecode files [16]. Every contract with a NLD less than 0.35 from some contracts of the sample will be classified as a Ponzi scheme. 55 new, potential Ponzi schemes were found with this measure technique. Only one collection was removed and the other 54 were classified as "hidden" Ponzi schemes. Altogether over 10% of all retrieved contracts contain this "illegal" system.

3.6.6. Ponzi Analysis

This section deals with analyzing the source code of different Ponzi schemes. Again, Azei et al. [3] provide important information for this topic. According to them, all found contracts
(discussed in the previous section 3.6.5) can be classified in one of four groups:

**Waterfall scheme:** If a new user joins, his investment will be divided between the already-joined users. Each user receives a fixed percentage of what he has invested.

**Handover scheme:** Only the address of the last user is saved. If a new user is interested in joining the system, a fixed amount has to be paid to the current stored address. Every turn increases the investment by a certain interest. Listing 4.3 shows a typical handover scheme. The condition in Line 13 is only true, if the new user sends enough money to the contract. This amount will be sent to the current address minus a fee (in this case 10%) for the contract and a new user and price are stored.

```solidity
contract HandoverPonzi {
    address owner;
    address public user;
    uint public
        price = 100 finney;

    function HandoverPonzi(){
        owner = msg.sender;
        user = msg.sender;
    }

    function(){
        if (msg.value < price) throw;
        user.send(msg.value * 9 /10);
        user = msg.address;
        price = price * 3 / 2);
    }

    function sweepCommission(uint amount){
        if (msg.sender == owner)
            owner.send(amount);
    }
}
```

Listing 3.8: Example of a handover scheme

**Tree-based pyramid schemes:** Has a data tree as structure and records every users address. On top stands the contract owner known as root. All other users are structured like a tree under the root. If someone joins the system, money will be split among its ancestors. If a
branch is not able to invite new users the bottom of it will not make any profit.

**array-based pyramid schemes:** Listing 3.9 shows a sample of this scheme. A user sends an amount of Ether to the contract and invokes the fallback function. A minimum fee of 1 Ether is required. If the user pays less, he will be rejected, otherwise he gets part of the array. The owner gets 10% of the investment and the rest goes to the other investors.

```solidity
contract ArrayPonzi{
    struct User{
        address addr;
        uint amount;
    }
    User[] public users;
    uint public paying = 0;
    address public owner;
    uint public totalUsers = 0;

    function ArrayPonzi() {
        owner = msg.sender;
    }

    function() {
        if (msg.value < 1 ether) throw;
        users[users.length] = User({
            addr: msg.sender,
            amount: msg.value});
        totalUsers += 1;
        owner.send(msg.value/10);
        while(this.balance >
            users[playing].amount *2){
            users[playing].addr.
            users[playing].amount *2); 
            paying+=1;
        }
    }
}
```

Listing 3.9: Example of an array-based pyramid scheme
3.6.7. Ponzi Statistics

Figure 3.4 shows the top 10 Ponzi schemes identified in the previous example. The second column lists the in- and out coming transactions of the contract. Also amount of Ether and USD is listed (note that the exchange rate accords to the day of the transaction). The most interesting part represents column Users. As you can see, most of the users will be left empty-handed. In only four contracts around half of the users have made some profit.

<table>
<thead>
<tr>
<th>Contract name</th>
<th># Trans.</th>
<th>ETH</th>
<th>USD</th>
<th>Users</th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynamicPyramid</td>
<td>418</td>
<td>143</td>
<td>7474</td>
<td>7437</td>
<td>83230</td>
</tr>
<tr>
<td>DianaEthereum-x1.8</td>
<td>277</td>
<td>166</td>
<td>5307</td>
<td>5303</td>
<td>63100</td>
</tr>
<tr>
<td>Doubler2</td>
<td>380</td>
<td>161</td>
<td>4858</td>
<td>4825</td>
<td>25432</td>
</tr>
<tr>
<td>ZeroPonzi</td>
<td>626</td>
<td>499</td>
<td>4490</td>
<td>4489</td>
<td>50867</td>
</tr>
<tr>
<td>Doubler</td>
<td>151</td>
<td>53</td>
<td>2977</td>
<td>2950</td>
<td>14251</td>
</tr>
<tr>
<td>Government</td>
<td>723</td>
<td>846</td>
<td>2938</td>
<td>2938</td>
<td>36311</td>
</tr>
<tr>
<td>Rubixi</td>
<td>624</td>
<td>61</td>
<td>1367</td>
<td>1363</td>
<td>16715</td>
</tr>
<tr>
<td>ProtectTheCastle1</td>
<td>765</td>
<td>757</td>
<td>1144</td>
<td>1138</td>
<td>12737</td>
</tr>
<tr>
<td>EthereumPyramid</td>
<td>965</td>
<td>338</td>
<td>986</td>
<td>917</td>
<td>4929</td>
</tr>
</tbody>
</table>

Figure 3.4.: Top 10 Ponzi schemes ordered by their Ether [3]

3.7. A Smart Contract in Detail

3.7.1. Structure of a simple Smart Contract

So far, our focus has been on smart contracts and ICOs superficial. It is time to take a deeper look at the structure and construction of a smart contract, according to an ICO. Listing 3.10 shows a small example, how to easily create a new token [10]. The used programming language of the Smart Contract is called Solidity. We need the keyword `pragma` to tell the compiler the Solidity version in which the smart contract has been written. `Contract Coin` can be seen as a structure, that contains functions and data. The type `address` is suitable for storing addresses of contracts. With the visibility `public`, the current value can also be accessed outside of the contract.

```
1  pragma solidity ^0.4.0;
2
3  contract Coin {
4    // The keyword "public" makes those variables
5    // readable from outside.
```
address public minter;
mapping (address => uint) public balances;

// Events allow light clients to react on
// changes efficiently.
event Sent(address from, address to, uint amount);

// This is the constructor whose code is
// run only when the contract is created.
function Coin() {
minter = msg.sender;
}

function mint(address receiver, uint amount) {
if (msg.sender != minter) return;
balances[receiver] += amount;
}

function send(address receiver, uint amount) {
if (balances[msg.sender] < amount) return;
balances[msg.sender] -= amount;
balances[receiver] += amount;
Sent(msg.sender, receiver, amount);
}

Listing 3.10: Sample how a ICO contract could look like

Next, we have the code `mapping (address => uint) public balance;` in line 7. Mappings have the same behavior as hash tables. A key exists and contains a value. Event `Sent(address from, address to, uint amount);` will be executed in the last line of the function `send`. After calling `Sent`, a listener get the parameters `from`, `to` and `amount`.

The function `mint` can only be called by the creator of the contract. The last function `send` can only be called by anyone who already has some coins on his balance.

### 3.7.2. Smart Contract of an ICO

For our example we take the smart contract of the ICO `adChain`. `AdChain` is a collaboration between `MetaX`, a blockchain technology company with focus on digital advertising, and `ConsenSys`, a production studio building dApps for the Ethereum blockchain [11].

The `adChain` Registry is a smart contract, which stores domains, that are rated as non-fraudulent by their `adChain` investors. `AdToken` is an essential component of the Registry. If a publisher prefers to be included in the registry, they have to make a deposit with their to-
ken and are a part of a challenge. In this challenge token holders can vote, whether they believe the publisher is a black sheep or not. As soon as the challenge time is up and the publisher wins, the domain of the publisher gets added to the smart contract on the blockchain and a part of the deposit will be rewarded to him. Otherwise the opposite is the case, the publisher does not get added to the registry and the challengers get a portion of the deposit awarded. An overview of this process can be seen in figure 3.5. Etherscan provides the published sale contract and at github you can see the whole source code.

The entire project contains seven solidity files including the main files sale.sol and token.sol. Sale.sol defines following ten storage variables:

- address public owner
- address public wallet
- HumanStandardToken public token
- uint public price
- uint public startBlock
- uint public freezeBlock
- bool public emergencyFlag = false
- bool public preSaleTokensDisbursed = false
- bool public foundersTokensDisbursed = false

---

8 https://etherscan.io/address/0x5678bcea6d6f33f645dca8e8c9b7d8d5ca0a2b1
9 https://github.com/AdChain/token-launch-contracts
• address[] public filters

Take a look at the two storage variables startBlock and freezeBlock. This is a main structure of every ICO smart contract. With the start storage we can prevent an early sale before it even has begun. The freezeBlock guarantees that the contract owner is not able to make any changes to the price and startBlock. So the final price is fixed before the sale and the investor knows the exact price for the token. To guard against some errors, an emergencyFlag exists. When this flag is set to true the sale stops from proceeding [13].

The sale contract contains nine functions. The most important and also vulnerable one is the purchaseToken function. This is the only function that can handle payments, thus, if a user makes a raw sending, it will be reverted immediately. The modifiers in the contract will cope such things. The modifier saleStarted reviews, if the current block number is higher than the start block. If not, the sale has not started yet and users are not able to interact with the contract. OnlyOwner checks, if the message sender is also the owner of the contract and notFrozen, if the current number is smaller than the current block number. Additionally, the emergency toggle and setup will be controlled.

```solidity
/// @dev purchaseToken(): function that exchanges ETH for ADT (main sale function)
/// @notice You're about to purchase the equivalent of 'msg.value' Wei in ADT tokens
function purchaseTokens()
    saleStarted
    payable
    setupComplete
    notInEmergency
{
    /* Calculate whether any of the msg.value needs to be returned to the sender. The tokenPurchase is the actual number of tokens which will be purchased once any excessAmount included in the msg.value is removed from the purchaseAmount. */
    uint excessAmount = msg.value % price;
    uint purchaseAmount = msg.value - excessAmount;
    uint tokenPurchase = purchaseAmount / price;

    // Cannot purchase more tokens than this contract has available to sell
    require(tokenPurchase <= token.balanceOf(this));

    // Return any excess msg.value
```
To get ADT, adChain tokens, a user has to call the \textit{purchaseToken} function. With the help of this function ETH will be exchanged to ADT. On line 13 in listing 3.11 the contract reviews, if the sent amount divided by the price will not include a remainder. If so the remaining Wei will be sent back to the user, who will be executed in line 22. The purchase can only be performed if there are enough adChain tokens remaining. This will be controlled in line 18. If not the contract should throw an exception and there is no possibility to give the contract some Ether. For sending the tokens to the wallet, \textit{transfer} is used due to its advantage over \textit{send}. If the transaction is out-of-gas the entire transaction will be reverted.
4. **Smartract CVA: a Smart Contract Creation, Validation and Analysing Tool**

The aim of this thesis was to develop a tool, that reviews a smart contract due to its functionality. As input we take a given smart contract of the ICO and transactions. The user has to take a look at the whitepaper and use the input field to give additional information. With this data we are able to check if the behavior equates to the promise of the contract.

The tool has been implemented with Flask, a Python framework. All required transactions must be provided by a CSV file or the address of the ICO. A small user interface has been designed for simplicity. It includes input fields for the smart contract address, transactions and an own created domain-specific language (DSL) for smart contracts. After this, the GUI displays if it has identified the functionality of the contract and if yes, does it really do what the whitepaper/the additional information says.

The whole execution can be seen as Black Box testing. The program compares the main programming language for smart contracts called Solidity. The given information via DSL and transactions has to be interpreted. Afterwards, a closer look at the smart contract and the transaction related to this contract will be explained. Some classifications are needed. After the collection of all the important data is finished, the comparison between the new created smart contract and given contract can begin. Through this tool people get an insight of the smart contract and can decide whether it is only scam or something valuable.

**4.1. Design and Architecture**

**4.1.1. Environment**

**Flask**

Flask is a powerful and easy to learn web framework for Python. The micro framework is based on *Jinja2* and *Werkzeug*. *Werkzeug* is used to manage software object requests, responses and utility functions. *Jinja* is a common Python template engine.
4 Smartract CVA: a Smart Contract Creation, Validation and Analysing Tool

**PostgreSQL**

For this project PostgreSQL is used as database. Flask commonly uses MongoDB, a non SQL database as storage.

**MPS**

For first experiments Jetbrains MPS was applied. It is a meta-programming language utilized to design domain-specific languages (DSL). MPS needs a projection engine since it applies projectional editing. The projection engine itself is built in Java for desktop applications. MPS is not recommended and suitable for web front-end solutions.

**TextX**

After being unlucky with MPS we tried TextX, a meta-language also for building domain-specific languages in Python. TextX is inspired by Xtext, a powerful framework and part of the Eclipse modeling project based on Java. With this meta-language we built our own textual language to express the meaning of whitepapers and generate a smart contract. The only thing required for TextX is a single language description.

**Porosity**

Porosity[18] is an open-source decompiler developed by Matt Suiche from Comae Technologies. This tool decompiles EVM bytecode back into readable Solidity syntax contracts. This reverse engineering tool made it possible to create smart contracts without the solidity code.

### 4.2. Smartract DSL

With the help of TextX we were able to create our own domain-specific language. In order to describe the context of a whitepaper a DSL is necessary. The created smart contract DSL called Smartract DSL is able to generate smart contracts that look similar to ICO contracts. Smartract DSL can be divided into two parts, the **Main Components** and their **Sub Components**. Each provide several keywords which help the user to create a smart contract. Following keywords are available:

- **Main Components**
  - `contract x [*]` ...initializes a smart contract where x defines the name
  - `variables x : y end`...variables where x defines the name, y the type

1https://pypi.org/project/textX/
4.3 Steps of Contract Validator

Following (three) steps have to be done (in order) to get a full analysis of a project:

- The first one confronts the user with general questions about the ICO project. Answering those questions will give Smartract CVA some information about trustworthiness referring to the venture.

- The second one uses an own created DSL called Smartract DSL. With the help of this DSL users can generate smart contracts. The user just has to read the whitepaper of the project and provides Smartract CVA some useful input.

- The last part presents the upload section. An original sale smart contract and also some transactions between the published contracts can be uploaded.

Details about the functionality and behavior of every single step will be explained in the next sub chapters.
4.3.1. Whitepaper

Getting a significant analysis of an ICO project requires some general information. This information will be inquired in the first step of the application. It confronts the user with six common questions about the Initial Coin Offering. To answer those questions, the operator has some given options, which can be simple as Yes, No and I don’t know.

Every single reply has a value. The value (or also points in this case that can be seen at the right of any choice) will be used to evaluate the selections of the user. The survey evaluation is designed as followed:

\[
_{ICO-RATING} = \sum_{x} Q_1 A + \frac{Q_2}{x} Q_2 A + \frac{Q_3}{x} Q_3 A + \frac{Q_4}{x} Q_4 A + \frac{Q_5}{x} Q_5 A + \frac{Q_6}{x} Q_6 A
\]

According to its importance in relation to ICO scams a weighting for each question is required where \( x = 100 \land \sum_{i=1}^{6} Q_i \equiv 100 \land Q_i \neq 0 \) is given.

To get a glimpse of the questionnaire, please take a look at the given structure below that lists all necessary queries and answers:

(Q₁) Does the project have a whitepaper?
(A1) Yes. [1p]
(A2) No. [0p]

(Q₂) Does the whitepaper promise a huge project that can be compared to Facebook, Instagram,…?
(A1) Yes. [0p]
(A2) No. [1p]

(Q₃) Is there already a launched product on the market?
(A1) Yes. [0p]
(A2) No, but it will be released soon. [0.5p]
(A3) No. [1p]

(Q₄) How do you rate the project team?
(A2) Huge experience in crypto assets. [1p]
(A3) Small experience in crypto assets. [0.5p]
(A4) Released projects with success (no cryptos). [0.5p]
4.3 Steps of Contract Validator

(A5) Failed with every project. [0p]

(Q6) Do they share updates with the community on a regular base?

(A1) Yes. [1p]

(A2) No. [0p]

(Q6) Can you find the source code on Github or other software development platforms?

(A2) Yes. [1p]

(A3) No. [0p]

4.3.2. Generate new Smart Contract

The next step of Smartract CVA confronts the user with Smartract DSL (for more details about the DSL take a look at chapter section 4.2). For the usage of Smartract DSL a whitepaper is required. Without a whitepaper of the project the user better ignores this part, except he wants to generate a random or user-specific smart contract.

The aim of step two is to generate a smart contract, which should be similar to the original one of the venture. First, the user has to take a look at the whitepaper in order to create such a contract. He must highlight important keywords and contents.

Next, the user tries to reproduce a smart contract with the help of Smartract DSL. Through the highlighted lines and keywords of the whitepaper a smart contract can be produced. This part may seem easy but it is definitely not. In fact it is not even possible to output the same smart contract as the ICO has, but you are able to come close to it. Smartract DSL creates contracts, which contain main parts of an Initial Coin Offering contract.

```solidity
contract Coin {
    variables
        minter : address
        balances : mapping
    end
    events
        Sent : event
        params
            from : address
            to : address
        amount : uint
```
functions
  Coin : function
    function_blocks
      assignment
        minter : eq : sender
      end
    end
  end
end

mint : function
  params
    receiver : address
    uint : amount
  end
  function_blocks
    if_condition
      sender : noteq : minter
    end
    assignment
      balancesreceiver : addeq : amount
    end
  end
end

send : function
  params
    receiver : address
    uint : amount
  end
  function_blocks
    if_condition
      balancessender : noteq : amount
    end
    assignment
      balancesmsgsender : subeq : amount
    end
    assignment
      balancesreceiver : addeq : amount
    end
  end
function_call
  Sent : function
    params
    sender : param
4.3 Steps of Contract Validator

After saving the input data, a smart contract (which can be similar to the provided venture) with the input DSL will be created. Listing 4.1 shows an input sample and Listing 4.2 the referring output.

```solidity
pragma solidity ^0.4.0;

contract Coin {
    /**
    * Storage
    */
    address public minter;
    mapping public balances;

    /**
    * Events
    */
    event Sent(address from, address to, uint amount);

    /**
    * Public functions
    */
```
function Coin()
{
    minter = sender;
}

function mint(address receiver, amount uint)
{
    if (sender != minter) return;
    balancesreceiver += amount;
}

function send(address receiver, amount uint)
{
    if (balancessender != amount) return;
    balancesmsgsender -= amount;
    balancesreceiver += amount;
    Sent(sender, receiver, amount);
}

Listing 4.2: Generated smart contract with the input provided in listing Listing 4.1

Let us take a closer at the provided code lines and begin with Listing 4.1. In line 1 we provide the contract name Coin and tell the DSL that we are going to create a contract. Inside this scope we are able to define different variables, functions and events. Every scope has to end with the keyword end, except the contract initializer contract. This one needs an opening and closing bracket.

In our shown example we define two variables in the variable scope. The left hand tells the DSL the identifier of the variable. On the right side we define a data type. In this case we have the address with the id minter and a mapping with the name balances.

Next we have the scope events at line 6. To define an event we need the structure [name of the event] : [type event] inside the scope. Likely as a function also an event can contain parameters. The parameters structure behaves like the variable structure.

Line 34 represents a function called send. A function contains params and function_blocks. Function_blocks offer different statements like an if_condition, for_loop and many more. In line 43 an assignment takes place. On the left we have the identifier again that will be assigned. In the center an operator is given, in this case subeq, which will be converted to -=. Version 1.01 of Smartract
4.3 Steps of Contract Validator

DSL supports operators directly, that means the input can be written as `!=` and not as a string like `noteq`.

Listing 4.2 displays the generated output of the just explained code. We do not discuss this part in detail.

### 4.3.3. Transactions/Reverse Engineering

The third and also final step of Smartract CVA takes a focus on transactions of the project. The old version of the application provided an upload interface for transactions, the whitepaper and original smart contract. Due to its simplicity we decided to remove the upload section and replaced it with an interface. In the new version (June 2019, v. 1.02) Smartract CVA uses the Etherscan.io API\(^2\). Etherscan’s Ethereum developer API supports GET and POST requests but has a limit of five requests every second. To use the API a free Key-Token has been created.

![Smartract CVA](image)

Figure 4.1.: Last step of Smartract CVA. Interface powered by Etherscan.io

Figure 4.1 shows the graphical user interface for the final step. Only the address as input is needed to get various information from Etherscan. Etherscan’s Ethereum developer API offers a lot of different modules. The most popular (and also most useful in our case) are accounts, contracts, and transactions. Five requests will be made to Etherscan:

- **[Module: Accounts]**
  - [Action: Balance] - Get Ether balance for the input address
  - [Action: TxList] - Get a list of normal transactions by the input address
  - [Action: TxListInternal] - Get a list of internal transactions by the input address

- **[Module: Contacts]**
  - [Action: GetSourceCode] - Get contract source code for verified contract source codes

\(^2\)https://etherscan.io/apis
4 Smartract CVA: a Smart Contract Creation, Validation and Analysing Tool

[Module: Stats]

[Action: EthPrice] - Get current price of Ether

After the successful http-calls, Smartract CVA is ready for analyzing all collected information and ready to give a rating to the venture.

4.3.4. Compare Results

After completing all previous mentioned steps the result can be viewed. The user has an option to download his new generated Smart Contract from step two with the provided DSL. This Smart Contract can be compared with the original one of the venture or also be used for a general comparison to ICO contracts.

The next section provides an overview of the entered address. The balance of Ether is listed and also converted into US dollar. The user can also see the total of all made transactions. An graphical illustration shows some details about the transactions.

![Result of Smartract CVA](image)

Figure 4.2.: Dummy Result of Step 3

At the end the venture gets a rating according to the answers in part one and the gathered information. Please take care that this rating is only a leaning, so Smartract CVA is not able to give the user a 100 percent guarantee if an ICO is serious about their project.
4.4. Behind the Application

In this subsection we do not go into detail about the whole code of Smartract CVA. We only analyze step two, the creation of a DSL and the interaction with it. Step one and three will be ignored due to its programming and logical simplicity. The application itself and the DSL grammar are written in Python with the help of Flask, a Python framework. For creating a DSL TextX\(^3\), a meta-language has been used. TextX is able to build automatically a meta-model and a parser for the created language. The framework Jinja2 acts as the template of a smart contract which means it has the same functional structure as a smart contract written in Solidity. Variable declarations, functions, loops and other programming parts are predefined in the template. The application needs some input, TextX produces a meta-model and placeholders in the Jinja2 template will be substituted with the created meta-model. As a result a smart contract in Solidity will be generated.

The Smartract DSL structure contains four files that will be explained in the next subsections. Following files are needed to build a meta-model, parse the created language and create a smart contract:

- **contract.ent** - Will be generated with the input of step two and represents the base of the contract
- **create_contract.py** - The heart and configuration file of Smartract CVA.
- **smart_contract.template** - Template file for a typical ICO smart contract
- **smart_contract.tx** - Includes grammar and all necessary rules to produce statements

4.4.1. Contract.ent

Listing 4.1 already shows an input example. With the input of the text-area in step two a new file will be created with the name *contract.ent* and the data will be copied to this file. This file builds the base for our smart contract. Due to our explanation in subsection 4.3.2 we do not focus on this file anymore.

4.4.2. CreateContract.py

*Create_contract.py* handles various configurations for the generation. We are importing *metamodel_from_file* from Textx and Jinja2. With *metamodel_from_file* we are able to use a metamodel from our textual description (in this case *smart_contract.tx*).

---

\(^3\)http://www.igordejanovic.net/textX/stable/
Jinja2 is a modern template engine for Python. With it, customization of tags, filters, tests and globals are possible. It is the default template engine for Flask. We initialize the template engine for our DSL with the following code:

```python
jinja_env = jinja2.Environment(
    loader=jinja2.FileSystemLoader(this_folder)
)
```

Listing 4.4: Initialization of jinja2

As previously mentioned, the operators in our textual description file have been identifiers in a previous version. For example `div` equals `/` or `mul` as `*`. A filter has been used for parsing the identifiers to their real meaning.

```python
def operator(op):
    return {
        'div' : '/',
        'eq' : '=",
        'mul' : '*",
        'sub' : '-",
        'add' : '+",
        'addeq' : '+=",
        'subeq' : '-=",
        'noteq' : '!=",
        'geq' : '>=",
        'seq' : '<=",
        'comp' : '==",
        'andand' : '&&",
        'oror' : '||",
        'and' : '&",
        'or' : '|",
        'greater' : '>',
        'smaller' : '<'
    }.get(op,op)
```

Listing 4.5: Different operators
After defining all different operators we put them to our Jinja environment.

```python
jinja_env.filters['operator'] = operator
```
Listing 4.6: Jinja Filter

Next we define our output folder where we store all our generated smart contracts. If the desired folder does not exist we are going to make it.

```python
srcgen_folder = join(
    this_folder, 'static/contracts'
)
```
Listing 4.7: Source output folder

```python
if not exists(srcgen_folder): mkdir(srcgen_folder)
```
Listing 4.8: Create output folder

After this it is time to load the Solidity template. The Solidity template will be explained in the next subchapter.

```python
template = jinja_env.get_template(
    'dsl_generation/smart_contract.template'
)
```
Listing 4.9: Create output folder

Now we are ready to build the model with the `contract.ent` file. To do this, we use the action `model_from_file` on our previous defined meta model variable.

```python
cntrct_m = cntrct_mm.model_from_file(
    join(this_folder,
        'dsl_generation/contract.ent'))
```
Listing 4.10: Create model from file
Now we have made all necessary steps and only the generation is left. To generate Solidity code we render the contract model into our Jinja template file and write it to a new file with the file ending .sol.

```python
# Generate Solidity Code
for contract in contract_model.contracts:
    # For each contract generate a solidity file
    with open(join(srcgen_folder, "%s.sol" % contract.name), 'w') as f:
        f.write(template.render(contract=contract))
```

Listing 4.11: Write solidity file

### 4.4.3. SmartContract.template

This subsection deals with the Jinja2 template. For each contract in our model we render one Solidity file with their statements. Usually templates have static and variable parts. The static parts will always be rendered as they are. Variable parts depend on the input, in our case on the defined model. In Jinja2 variable parts are written inside two brackets. In the inline code we define a variable part for the contract name.

`contract {{contract.name}}
```

Listing 4.12: Declaration of a contract

After initializing the contract with a name, various conditions are set. The first one checks if the model has global variables set. If so, every single variable will be set (in our template global variables are set public as default and represent a static part example). Normally a tab is set in a for loop but in this case we do not need one. Our global variables should be set at the right spot.

```jinja
{% for variable in contract.variables %}
{{variable.type}} public {{variable.name}};
{% endfor %}
```

Listing 4.13: Print global variables

Next, we produce all events from the contract model. Only the keyword `event` is static due to this keyword always remains the same. An event contains a name, followed by parameters. After every parameter a comma is set. The last parameter is closed with bracket and semicolon.

```jinja
{% for event in contract.events %}
  event {{event.name}}({% for param in event.params %}{% for param in event.params %}{param.type}{{param.type}}{% endfor %}
```

Next, we produce all events from the contract model. Only the keyword `event` is static due to this keyword always remains the same. An event contains a name, followed by parameters. After every parameter a comma is set. The last parameter is closed with bracket and semicolon.

```jinja
event {{event.name}}({% for param in event.params %}{% for param in event.params %}{param.type}{{param.type}}{% endfor %}
```
Function modifiers are used to change the behavior of a function in an easy way. Before a function executes a modifier can automatically check a specific condition. An important side note: modifiers are inheritable parts of a contract. The template structure for modifiers looks as follow:

```solidity
{% for modifier in contract.modifiers %}
  {{modifier.type}} {{modifier.name}}{
    {% for block in modifier.function_blocks %}
      {% if block.assert_statement %}
        assert({{block.assert_statement.left_cond}}
        {{block.assert_statement.operator|operator}}
        {{block.assert_statement.right_cond}});
      {% endif %}
      {% if block.require_statement %}
        require({{block.require_statement.left_cond}}
        {{block.require_statement.operator|operator}}
        {{block.require_statement.right_cond}});
      {% endif %}
    {% endfor %}
    _;
  }
{% endfor %}
```

Listing 4.15: Modifiers structures

More information about modifiers and some examples can be found at the Solidity documentation.

The last part in our template cares about functions. This one is the most complex coding section and has some limitations. A function structure contains a name, parameters and a function block. Inside a function block we are able to build a lot of different statements.

One of the limitations mentioned before is the loop statement. A for-loop is no problem at all. A nested for-loop in a for-loop is also easy for our Smartract DSL but if we have more than two-nested for-loops we are not able to represent this smart contract (version 1.1). Maybe it seems odd but in general a smart contract of an ICO does not have two nested loops (or should not). Every execution costs gas and so a good contract can avoid such structure in most cases.

---

4 https://solidity.readthedocs.io/en/v0.4.24/contracts.html
and maintain gas low. Another limitation referring loops is the possibility of while-loops. In Smartract DSL version 1.1 we are only able to generate for-loops. While-loops will follow.

### 4.4.4. SmartContract.txt

We already defined a model file, a configuration file and a template file. The only file left is a grammar file with rules. *Smart_contract.txt* holds the grammar rules according to the smart contract generation. subsection 4.4.4 displays the connection of the grammar of Smartract DSL.

![Diagram of grammar rules](image)

Figure 4.3.: Grammar rules of Smartract DSL

The basic building blocks of the description language are rules. The *ContractModel* represents the base of every contract.

**ContractModel:**

```plaintext
contracts+=Contract
```

Listing 4.16: ContractModel

The rule in *ContractModel* tells us the pattern of the contract object. The contract object body contains a fixed string 'contract' followed by an input name. When building a model, the DSL looks for the fixed string 'contract' and knows, that the string followed is the name of the contract. Brackets serve as delimiter for the body rules. Inside the body scope we have four different rules: *variables, events, modifiers, and functions*.

**Contract:**
4.4 Behind the Application

Listing 4.17: Contract rules

Variables have a simple rule. They only need an identifier as a name and a datatype. An event has an additional rule to allow parameters. Functions have the same rules but also add another one. To build the main part of a function a rule called function_blocks is given. The object of a modifier looks similar without allowing parameters:

Modifier:

Listing 4.18: Modifier rule

The focus of modifiers and functions lies on the main rule called function_block. Inside the mentioned scope another seven rules can be built. Below shows the structure of a function_block.

Function_block:

Listing 4.19: Function block rules

As already mentioned a fixed string serves as orientation for Smartract DSL. The assignment rule has two options. We can have an assignment were we need the datatype for the left hand
side and one without. To do this we simply use an or operator. Smartract DSL knows which option it should pick due to the number of given arguments (three vs. four).

Assignment:

\[
(\text{left}_\text{assign}=\text{Datatype} ':', \text{operator}=\text{Operators} ':', \\
\text{right}_\text{assign}=\text{Datatype}) \mid (\text{type}=\text{Datatype} ':', \\
\text{left}_\text{assign}=\text{Datatype} ':', \text{operator}=\text{Operators} ':', \\
\text{right}_\text{assign}=\text{Datatype})
\]

Listing 4.20: Assignment rules

Next we define the condition rule if\_condition. In version 1.0 only the condition itself is defined with a return. We made a simple return definition because we took a look at over 30 crowd-funding smart contract samples and they all had an if condition like this. Version 1.1 contains a scope for a given condition. The old condition:

If\_condition:

\[
\text{left}_\text{cond}=\text{Datatype} ':', \text{operator}=\text{ID} ':', \text{right}_\text{cond}=\text{Datatype}
\]

Listing 4.21: If-condition rule

And the new one:

If\_condition:

\[
\text{left}_\text{cond}=\text{Datatype} ':', \text{operator}=\text{ID} ':', \text{right}_\text{cond}=\text{Datatype} \\
('\text{condition\_block' ifblock = If\_block 'end}')
\]

Listing 4.22: Declaration of a contract

There is also a possibility to make a function\_call inside a function\_block. A function\_call has the same structure as the previous shown event.

A more interesting rule displays the for\_loop. Inside the scope there are four rules. First one defines, where we start the from iteration, second one when the loop will hold, third one holds the incrementor and last but not least a function\_block. But why a function\_block? Inside a loop we can have all the same rules like a function. To take care of redundancy we did not create a now grammar block.

For\_loop:

\[
'\text{from'} from\_loop = \text{Assignment} 'end' \\
'till' till\_loop = \text{Assignment} 'end' \\
in\text{c}=\text{STRING} \\
('\text{function\_blocks' function\_blocks += Function\_block 'end'})
\]
Above an excerpt of the for Loop from smart_contract.tx has been listed. To build the model from a contract with the mentioned for Loop a DSL file contract.ent like the following is needed:

```solidity
    distributePreBuyersReward : function
        params
            _preBuyers : address
            _preBuyersTokens : uint
        end
        function_blocks
            assert
                "!preSaleTokensDisbursed"
            end
            for_loop
                from
                    i : = : 0
                end
                till
                    i : < : "_preBuyers.length"
                end
                "i++"
            function_blocks
                require
                    "token.transfer(_preBuyers[i],_preBuyersTokens[i])"
                end
            function_call
        TransferredPreBuyersReward : function
            params
                _preBuyers : param
                _preBuyersTokens : param
            end
        end
        end
        assignment
            preSaleTokensDisbursed : = : true
        end
    end
```

Listing 4.24: Small DSL example
The whole abstract represents a function called `distributePreBuyersReward`. From line two till four the parameters section is defined. With line six the main part, `function_blocks` begins. Let’s jump to line ten were we can find the rule for building a loop. The statements `from` and `till` are only simple assignments and need an incrementor, that is a single string. After this in line 18 a `function_blocks` inside a `for_loop` has been created. Inside this scope we can create several, different rules again.

If we run our script directly or with Smartract CVA we can expect the following results according to the `distributePreBuyersReward` function:

```solidity
function distributePreBuyersReward(
    address _preBuyers,
    uint _preBuyersTokens)
{
    assert(!preSaleTokensDisbursedNone);
    for(i = 0, i < _preBuyers.length, i++){
        require(token.transfer(_preBuyers[i],
            _preBuyersTokens[i]), None None);
        Transferredprebuyersreward(_preBuyers,
            _preBuyersTokens);
    }
    preSaleTokensDisbursed = true ;
}
```

Listing 4.25: Function distributePreBuyersReward

Next to the previous described `for_loop` only `return`, `assert` and `require` are left. `Return` returns an identifier followed by a semicolon\(^5\). The assert and require statements have the same rule building structure that looks like this:

```
Require_statement:
    (left_cond=Datatype '::'
        operator=Operators '::'
        right_cond=Datatype) |
    (left_cond=Datatype)
;
```

Listing 4.26: Require statement

With the help of the `or` operator and TextX, rules can be built with three or only one parameter specified. Smartract DSL only supports four data types in the current version (v1.2): integer, double, float and strings.

\(^5\)semicolon is set inside the template
5. Evaluation

The aim has been to show and find different vulnerabilities referring to smart contracts. In this relation a small smart contract tool has been developed and helps evaluating an ICO, whether it fulfills its conditions or only represents a scam. Questioning (ICO general information), engineering (Smart Contract DSL) and analyzing (Etherscan.IO) are the three required steps in order to achieve a successful estimation. Each part can be crucial for an ICO trustworthiness. If one fails, the tool recommends operators to keep out of the ventures business due to less information.

In order to evaluate an ICO with the validator tool, a critically examination of the developed tool Smartract CVA is needed. For the evaluation, two different methods are used. A detailed analysis of the particular methods is presented in the following. Only the most relevant findings are discussed. The first subchapter deals with the criteria-based software evaluation of Smartract CVA. Many different parts like

- analyzing and collecting different information about the program’s behavior,
- performance,
- meaningfulness
- and outcome

are analysed.

The second subchapter uses the Technology Acceptance Model (TAM). A small amount of users had to test the Smartract CVA tool on specific criteria and complete a survey before and afterwards.

The last subchapter summarizes and combines all different methods and deals with previous stated research questions. The purpose of this assessment is to improve effectiveness and future programming directions.
5.1. Criteria-based Software Evaluation

The evaluation is divided into five parts to classify this thesis. One subsection covers the whole research and theory. For the application Smartract CVA itself we are using another four different subsections and reviewing following categories and criteria: functionality, usability, performance and documentation.

5.1.1. Evaluation: Research

In this study we explained smart contracts in detail, gave a short overview of decentralized systems and Ethereum, showed different criminal actions, fraud schemes of initial coin offerings and developed a tool for creating, validating and analyzing smart contracts. To take care of the main topic smart contracts we had to give a brief explanation about Ethereum and cryptocurrency. Also some information about ICOs were needed to understand the behavior and benefit of smart contracts. After getting a base knowledge about the digital currency we delved deeper into the topic and took a look at the basic weaknesses of smart contracts.

Collection of data

The basic steps for planning a study are still relevant to get useful information. The starting point for this paper was Google Scholar which delivered around five useful papers containing the keywords smart contracts, Ethereum, cryptocurrencies and ICOs. To collect data according to this topic a snowballing technique[23] has been used. Backward snowballing, the first used research approach takes the literature references from a starting point to spot new papers. Useless papers that are not fulfilling basic criteria such as keywords, authors and publication year from the bibliography will be excluded for the future research. Also already examined literature will be removed. After removing every useless literature in the starting point a good base for inclusion is given with the remaining papers. Most relevant information of the remaining papers will be collected and afterwards a look at the literature list of the remaining can bring new potential papers for inclusion. Forward snowballing defines the opposite of backward snowballing. With this technique new papers are identified based on papers that are citing a paper of the starting point.

Do find useful analysis data different blockchain explorers like Etherchain, Etherscan and blockchain.com have been applied. Each of them has some advantages and disadvantages like Etherscan, which offers an API for the Ethereum blockchain but only allows five requests per second.
5.1 Criteria-based Software Evaluation

Investigation

The purpose of this thesis was to show various, malicious smart contracts of ICOs and how to prevent such investments. To accomplish this condition we made a detailed research about different attacks like the DAO and developed Smartract CVA, a smart contract tool. This thesis did not show any self-made experiments in relation to smart contracts due to the focus and development of a tool.

5.1.2. Evaluation: Functionality

The first part of the criteria-based assessment covers functionality. Smartract CVA provides a useful and time-saving solution for creating simple, secure smart contracts. Current ICOs often turn out to be scam and do not aim to achieve their promised project. With the developed tool users get a small rating over the desired ICO and can decide on their own if they are willing to invest.

To guarantee the above mentioned functionalities we used a data set of over 20 different ICOs. All three steps of the evaluation application have been made with several ICOs so we were able to compare the created smart contracts with the original ones. A direct comparison of a smart contract is not that easy as it seems in first place. More about strength and weaknesses relating to Smartract CVA will be presented in 5.1.6.

The user interface explains effectively all required data and describes how to use the tool with an example as introduction. The introduction sample should solve all obscurities.

5.1.3. Evaluation: Usability

As previously mentioned a user friendly interface without any complicated required input data is used. The start page gives a short description of the tool, how it works and what steps are waiting for the user.

The first step confronts the user with simple questions about the ICO. To answer the questions a dropdown with different choices is given. All choices and questions are clear defined and simple to understand. If a question is not easy to answer, the user has the opportunity to answer with i.e. ”I don’t know”.

The next and second step is about creating a similar Smart contract relating to the ICO. This step is the most tricky one but with the provided sample the user should be able to handle Smartract DSL. All available rules to build the contract are displayed above the input text.

\[^{1}\text{Smartract CVA is not able to give a 100\% guarantee referring to ICO}\]
area. The text area itself contains a sample of a mini contract as placeholder.

To build the DSL a whitepaper is needed. The user should go through the whole whitepaper and mark important keywords and maybe also hints about the programming architecture. Next this keywords can be used to define rules within Smartract DSL. To build similar contracts as the ICO some background knowledge about initial coin offerings related to smart contracts design and architecture can be useful. If the user is not sure about how to build the contract he can ignore step two and forward to step three.

The last and third step deals with the Etherscan.io interface. With the provided API of Etherscan Smartract CVA collects useful data related to the provided contract address. In other words, the last step contains only one text field as input which needs the address of the ICO. To locate the address of a venture explorers like Etherscan and Etherchain can be helpful.

5.1.4. Evaluation: Performance

Due to no database queries and only five simple GET request to etherscan.io the performance is very satisfying. Smartract CVA does not need any optimizations related to performance. No exact time measurements have been made because of futility.\(^2\)

5.1.5. Evaluation: Documentation

In performing the evaluation of the documentation\(^3\) we consider a criteria-based assessment provided by [15]. Table 5.1 shows a detailed scoring of the Smartract CVA documentation where a scale of one (as its minimum) to ten (as its maximum) is used.

5.1.6. Strength and Weaknesses

The evaluation and review of this research and the developed tool Smartract CVA revealed numerous strengths and weaknesses. Different types of attacks and an application for various purposes related to smart contracts have been the focus of this thesis. This section highlights advantages and disadvantages.

Strength

One of the strength points of this paper is the useful listing of different types of malicious attacks referring to initial coin offerings. One major example shows the famous Ponzi scheme that is also a common thing in today’s smart contract programming. The Ponzi scheme can be

\(^2\)Requests are under a second
\(^3\)Can also be seen as a sub-criterion of usability
### 5.1 Criteria-based Software Evaluation

<table>
<thead>
<tr>
<th>Documentation</th>
<th>Comment</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides a high-level overview of the software</td>
<td>This thesis gives a detailed overview about <em>Smartract CVA</em> and the related DSL <em>Smartract DSL</em>. Front- and back-end are explained blow-by-blow.</td>
<td>10</td>
</tr>
<tr>
<td>Partitioned into sections for users and developers</td>
<td>A partition between users and developers is not set. In the provided README at the GitHub repository a short explanation states how to run the application as a user or developer locally. Developers can add or improve the existing application as they like. A documentation for developing is only given inside the source code via comments. In sum there is not a clear division/section between users and developers.</td>
<td>5</td>
</tr>
<tr>
<td>Lists resources for further information</td>
<td>The application Smartract CVA does not reference a source for further information. To get further information about the tool a glance at this thesis is needed.</td>
<td>3</td>
</tr>
<tr>
<td>Consists of clear, step-by-step instructions</td>
<td>Smartract CVA is divided into three steps. The first step is a questionnaire. The second affects Smartract CVA and the last one contains a API for etherscan.io. In each part a clear guide shows the user how to deal with the current step. Also an instruction example with all three steps is given. To get more information, a look at the paper can be helpful.</td>
<td>10</td>
</tr>
<tr>
<td>Gives examples of what the user can see at each step e.g. screen shots or command line excerpts</td>
<td>Screenshots or excerpts of the result are not given directly at the interface of each step. A whole result can be seen in the introduction sample which contains plots about an example result.</td>
<td>8</td>
</tr>
<tr>
<td>States command names and syntax, says what parameters to use</td>
<td>Commands and syntax is only required in step two. In this step a example is given trough the instruction sample and also a mini contract inside the text area as placeholder. The <em>Smartract DSL</em> syntax is present over the input field and states every single possible parameter.</td>
<td>10</td>
</tr>
<tr>
<td>Documentation on the project web site makes it clear what version of the software the documentation applies to</td>
<td>The documentation belongs to the current version number and will be updated if any major changes are made.</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 5.1.: Documentation evaluation of Smartract CVA according to [15]
divided in four different types (waterfall, handover, array-based pyramid and tree-based pyramid schemes). Each has been described in this paper and also some examples have been shown.

Another strength point of Smartract CVA is that it is able to create smart contracts based on the user input. With this opportunity users without any programming skills can create Solidity files by using the developed description language Smartract DSL.

But that is not the only value of Smartract CVA. The application is also able to give an ICO a rating if it is a scam or maybe deserves some trust. To get a useful result the user has to answer some basic questions about the venture. If he is skilled enough with Solidity, he should take a look at the generated code and the original one.

A graphical overview of transactions, Ether in and outcome, number of transactions and other data is given with the application programming interface offered by etherscan.io.

**Weakness and Opportunities**

Smartract CVA has a lot of positive functions and can be useful for investors, project creators and developers. But as every application also Smartract CVA has some weaknesses.

One of the main weaknesses (or a missing feature) is a direct comparison between the created and original smart contract. The creation of a smart contract can be really useful, but a tool which is able to compare both would be peak. This could be a future research and improvement topic but will be tricky. A possibility may be to compare directly at transaction and not at source code level. Each transaction of the created smart contract should have the same output as the original smart contract. This kind of solution can be tried manually, but is not implemented in Smartract CVA. Another suggestion was to try the reverse engineering way. With the help of Porosity, a smart contract decompiler, already published contracts can be decompiled to the EVM level. With the help of compiling the created contract to EVM level, we would be able to compare already published once on the same level. More about tries and limitations in Section 6.1.

Another weakness of the developed program is the created domain-specific language itself. Nested loops and functions are only possible till a specific level. This may not be a problem due to most serious crowdsale smart contracts avoid too much nested loops, but our research showed, that there also exist some smart contracts with deeper loops than the DSL is able to handle.

There also exist smart contracts with commands that are not supported by Smartract DSL. It is able to build all basic and important rules but of course the description language is also
limited. Each function, loop and other components can be built with Smartract DSL but maybe not in the same constellation. Objects are not supported by the DSL in version 1.1.

Last, smart contracts are not as simple as shown in our tool. In our tool a crowdsale only consists of one file. In real world most contracts include many other Solidity files and create their own token structure.

5.2. Technology Acceptance Model

In this subchapter we propose a test model based on the technology acceptance model (TAM). In addition a questionnaire had been developed to gather information of a small appliers group. The survey is split into three parts:

- General information (Skills and experience with ICOs)
- The tool Smartract CVA
- Technology acceptance questionnaire

The results reveal, that Smartract CVA can help both, experienced and non experienced users, to investigate ICOs and create simple smart contracts.

5.2.1. Overview of the Technology Acceptance Model (TAM)

The technology acceptance model (TAM), introduced by Davis [8], is used to predict user acceptance of any information technology. Design problems can be solved before any user uses the system with two different factors: perceived usefulness (PU) and perceived ease of use (PEU). Perceived usefulness is defined as a scale to which an applier believes, using a particular system would enhance the task performance. Factor perceived ease of use is described as the degree to that a person believes that using a distinct information technology system would be free from effort. Altogether five components are included in the first created TAM. Beside perceived usefulness and perceived ease of use also attitude toward using (ATU), behavioral intention to use (BI) and actual system to use are included. ATU states the preprocess of BI and directly foretells the behavioral intention of the user. Figure 5.1 shows a replication of the original technology acceptance model by Davis in 1989.
5.2.2. Method

Instrument - Questionnaire

For the purpose of this study, a questionnaire with the help of Google forms\(^4\) including three different main parts has been drawn up. The final survey included 5 general information questions about programming skills and blockchain experience (part one), 10 items related to Smartract CVA (part two) and 18 items for the technology acceptance model (part three). The previous stated model (figure 5.1) has been taken as a reference for the third part. In this part, four different factors are included and serve as a measure for the TAM. Those factors are:

- Perceived usefulness (PU) with five items
- Perceived ease of use (PEU) with six items
- Behavioral intention to use (BI) with three items
- Attitude toward using (ATU) with four items

Part one and two contain question with simple answers or multiple choices. Part three offers a seven-point style range from 1 (strongly disagree) to 7 (strongly agree). A detailed look of the questionnaire is given in A.1

5.2.3. Context and Validity

For testing the developed tool Smartract CVA (and completing the questionnaire), an Amazon EC2 instance has been purchased. With the acquired instance of Amazon we were able to host our application and provide it for a limited time to our testers. Experienced testers also had the possibility to install Smartract CVA on their own with an GitHub repository.

The validation of content was established on the opinion of two experts in the blockchain

\(^4\)https://www.google.com/forms/about/
section with advanced experience in programming smart contracts. Altogether 11 people (including the two experts) took part at the questionnaire and tested the Smartract CVA tool, but only 9 of them had been declared to be valid. Two questionnaire showed anomalies to their statements, so we did not count them. The results of part one 5.2 and two had been measured with a sample percentage of each answer. For part three we used Cronbach’s alphas [19], which is a common used measure referring to the technology acceptance model.

5.2.4. Results and Discussion of the Questionnaire

Part I and II - General Information and Smartract CVA

Descriptive tables and statistics were conducted to show different answers and aspects of the users. Table 5.2 summarizes the score for each question in part one.

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer 1</th>
<th>Answer 2</th>
<th>Answer 3</th>
<th>Answer 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming skills</td>
<td>0 (0%)</td>
<td>1 (11%)</td>
<td>3 (33%)</td>
<td>5 (56%)</td>
</tr>
<tr>
<td>Blockchain experience</td>
<td>2 (22%)</td>
<td>1 (11%)</td>
<td>3 (33%)</td>
<td>3 (33%)</td>
</tr>
<tr>
<td>Solidity experience</td>
<td>6 (67%)</td>
<td>0 (0%)</td>
<td>1 (11%)</td>
<td>2 (22%)</td>
</tr>
<tr>
<td>Have you ever took part at an ICO crowdsale?</td>
<td>3 (33%)</td>
<td>4 (44%)</td>
<td>2 (22%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>If yes, did you receive any cashout?</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
<td>2 (22%)</td>
<td>7 (78%)</td>
</tr>
</tbody>
</table>

Table 5.2.: Questionnaire evaluation of part one

Six users describe themselves with good or excellent skills according to the blockchain technology. Around two of them also contain excellent experience in Solidity programming. Most (over 50%) do not have any basics with Solidity. Concerning the investment in ICOs, only two users spend mony into an initial coin offering. Those users good some money back but nobody made a profit with it. They all received less than their deposit.

In part two, seven users (78%) specified an ICO. Two left them blank. Four (57%) of them took the ICO adChain (ADT). The reason for this may be that smart contract samples of three different initial coin offerings are given and one of them is adChain. The other three ICOs were Storm, basic-attention-token (BAT) and Angor. Nobody stated any problem with step one related to the web application Smartract CVA. Most of them stated it was easy to only select an given item and to make research about an ICO. More people had problems with part two. All of answered, that they used given examples to create a smart contract with the Smartract DSL. Seven users noted, that they had no (or little) problems with step three. Six testers crossed yes for the question Did Smartract CVA help you in your decision?. Table 5.3 shows
the result of all remaining questions with a ten-point evaluation system from 1 (too difficult) to 10 (too easy)\textsuperscript{5}.

<table>
<thead>
<tr>
<th>Question</th>
<th>Total points</th>
<th>Average points</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>On a scale of 1 to 10 - how easy was it for you to complete the survey in step number I?</td>
<td>78</td>
<td>8.6</td>
<td>really easy</td>
</tr>
<tr>
<td>On a scale of 1 to 10 - how easy was it for you to recreate a smart contract?</td>
<td>54</td>
<td>6</td>
<td>neither easy nor difficult</td>
</tr>
<tr>
<td>On a scale of 1 to 10 - how easy was it for you to enter the right smart contract address?</td>
<td>62</td>
<td>6.8</td>
<td>rather easy</td>
</tr>
<tr>
<td>How good was your recreation of the original crowdsale smart contract in step II?</td>
<td>51</td>
<td>5.7</td>
<td>some parts were identical</td>
</tr>
</tbody>
</table>

Table 5.3.: Questionnaire evaluation of part two

**Part III - Technology Acceptance Model**

Part three of the survey deals with questions referring to the technology acceptance model. Perceived usefulness, perceived ease of use, behavioral intention and attitudes towards creating a smart contract have been determined. Table 5.4 summarizes the mean scores and the standard derivation of all participants on the five different questions on PU. The mean score has also been summed up for experts only. As we can see, experts gave a higher rating in general. Only PU5 has a better rating in general. According to our participants Smartract CVA scores for creating smart contract but is not as strong at analyzing them.

\textsuperscript{5}For question 13, 1 defines *Not able to recreate* and 10 defines *Identical*
<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean (all)</th>
<th>Mean (exp)</th>
<th>SD (all)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU1</td>
<td>4.2</td>
<td>5.5</td>
<td>1.4</td>
<td>Smartract CVA saves time compared to own detailed research about an ICO</td>
</tr>
<tr>
<td>PU2</td>
<td>5.33</td>
<td>6.5</td>
<td>0.68</td>
<td>Smartract CVA can be used to create smart contracts</td>
</tr>
<tr>
<td>PU3</td>
<td>3.6</td>
<td>4</td>
<td>1.03</td>
<td>Smartract CVA can be used to validate smart contracts</td>
</tr>
<tr>
<td>PU4</td>
<td>3.55</td>
<td>4</td>
<td>1.06</td>
<td>Smartract CVA can be used to analyze smart contracts</td>
</tr>
<tr>
<td>PU5</td>
<td>4.67</td>
<td>4</td>
<td>0.73</td>
<td>Smartract CVA allows me to create smart contracts quickly</td>
</tr>
</tbody>
</table>

Table 5.4: Perceived Usefulness (PU) - mean score of all participants/only experts of each item in the questionnaire. Additionally a standard deviation is given.

Table 5.5 deals with the perceived ease of use. Nobody had problems with step one in Smartract CVA. They were all able to answer simple questions in the offered tool. Step two (creating a smart contract) has been challenging for them. Most of the users were able to manage step two with the offered template, but described it as tricky when they had to build a smart contract on their own. Step three (searching and entering the smart contract address) has been easy for most of the participants. Some stated, that they did not know which address they have to use. Those users quoted, that they only have small or no knowledge with smart contracts.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean (all)</th>
<th>Mean (exp)</th>
<th>SD (all)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEU1</td>
<td>4.2</td>
<td>5</td>
<td>1.4</td>
<td>Using Smartract CVA is easy for me</td>
</tr>
<tr>
<td>PEU2</td>
<td>6.33</td>
<td>6.5</td>
<td>0.22</td>
<td>Complete step one of Smartract CVA is easy for me</td>
</tr>
<tr>
<td>PEU3</td>
<td>4.11</td>
<td>4</td>
<td>1.32</td>
<td>Creating a smart contract (step two) with Smartract CVA is easy for me</td>
</tr>
<tr>
<td>PEU3A</td>
<td>3.11</td>
<td>4</td>
<td>1.08</td>
<td>Creating a smart contract (step two) without an offered template is easy for me in Smartract CVA</td>
</tr>
<tr>
<td>PEU4</td>
<td>4.44</td>
<td>4</td>
<td>1.43</td>
<td>Complete step three of Smartract CVA is easy for me</td>
</tr>
<tr>
<td>PEU5</td>
<td>3.44</td>
<td>3</td>
<td>1.11</td>
<td>I do not have any problems building a DSL in Smartract CVA on my own (without Samples)</td>
</tr>
</tbody>
</table>

Table 5.5.: Result of perceived ease of use

According to table 5.6 some users stated, that they will use Smartract CVA in the future. Others claimed that they prefer research on their own and do not want to use any tool. “The analyzing part of Smartract CVA is good, but should be improved and can not compete with other big players like etherscan.io”, a participant wrote.
5.2 Technology Acceptance Model

### Table 5.6.: Result of behavioral intention to use

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean (all)</th>
<th>Mean (exp)</th>
<th>SD (all)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI1</td>
<td>3.44</td>
<td>4</td>
<td>1.34</td>
<td>I will try to use Smartract CVA instead of any other research method about an ICO</td>
</tr>
<tr>
<td>BI2</td>
<td>3.11</td>
<td>3.5</td>
<td>1.46</td>
<td>I will create smart contracts with Smartract CVA</td>
</tr>
<tr>
<td>BI3</td>
<td>2.89</td>
<td>3.5</td>
<td>1.6</td>
<td>I will analyze ICOs with Smartract CVA</td>
</tr>
</tbody>
</table>

Unexperienced users had problems with using the tool. Experts could be convinced by Smartract CVA and see potential in future development if some new features and improvements will be offered.

### Table 5.7.: Result of attitude toward using

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean (all)</th>
<th>Mean (exp)</th>
<th>SD (all)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATU1</td>
<td>3.44</td>
<td>4.5</td>
<td>1.26</td>
<td>I prefer using Smartract CVA over any other related tool</td>
</tr>
<tr>
<td>ATU2</td>
<td>4.33</td>
<td>5</td>
<td>0.92</td>
<td>I prefer using Smartract CVA contract validator over an other related tool</td>
</tr>
<tr>
<td>ATU3</td>
<td>2.89</td>
<td>3.5</td>
<td>0.87</td>
<td>I prefer using Smartract CVA contract creator over other related tools</td>
</tr>
<tr>
<td>ATU4</td>
<td>3</td>
<td>3</td>
<td>0.73</td>
<td>I prefer using Smartract CVA contract analyzer over other related tools</td>
</tr>
</tbody>
</table>

Overall every single section of the technology acceptance framework received an acceptable Cronbach alpha score (table 5.8). Only ATU has been under 0.7 and should be questioned. Cronbach alpha shows how well a questionnaire, survey or other test method measure how it
should. A higher reliability shows a better measurement. Low reliability means the measures are not as significant as they should be. A good score has been made with PEU which received over 0.82.

<table>
<thead>
<tr>
<th>Measurement tool</th>
<th>Number of items</th>
<th>Cronbach alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceived usefulness</td>
<td>5 items</td>
<td>0.7668</td>
</tr>
<tr>
<td>Perceived ease of use</td>
<td>6 items</td>
<td>0.8201</td>
</tr>
<tr>
<td>Behavioral intention of use</td>
<td>3 items</td>
<td>0.7095</td>
</tr>
<tr>
<td>Attitude toward using</td>
<td>4 items</td>
<td>0.6686</td>
</tr>
</tbody>
</table>

Table 5.8.: Cronbach alpha coefficient value for all 18 items in part III. Calculated with [21]

5.3. Reflection on Research Questions

In this section we are facing the result of the three stated research questions at the beginning in section 1.3. Each subsection demonstrates a research question and the associated output in a short discussion.

What are common used fraud systems referring to smart contracts and how do they affect reputation and crowdsales of ICOs?

In section 3.4.3 different types of fraud systems have been explained and stated the answer to this RQ. There are simple attacks like the DAO that had a fallback-loop issue where users were able to steal Ether without limits. Also fraud systems like Ponzi are common in todays cryptocurrency especially in initial coin offerings. Due to many malicious circumstances ICOs gained a bad reputation in public. Investors are more careful about investing in a crowdsale. Recent studies from icorating.com\(^6\) showed that Q3 2018 only gathered 1.8 million us-dollars. For comparison: Q3 in 2017 made over 8 million dollars. The reason of less investments are intensive obligations especially from the American tax authority SEC and losing trust in dubious projects.

Is there a possibility to really guarantee that an ICO is doing what they promise? If yes, how can we identify and prevent them?

There is no possibility to be certain if someone can trust the project of an ICO, but various frameworks and tools can help to prevent fraud systems. In section 3.4.2 simple steps are shown how users can rate an ICO with the help of a specific framework. Users have to perform five steps like:

\(^{6}\)https://icorating.com/report/ico-market-research-q3-2018/
5.3 Reflection on Research Questions

- Reviewing the whitepaper
- Look at the team behind the project
- Check if a community exists
- Check, if they share an open-source code
- Check token distribution and value

Empirical analysis consider the financial parts of ICOs. They gather a set of control variables like the initial price of a token, current price of a token and the total returns. In 2018 a new platform called icorating\(^7\) shares reviews about new ICO. They determine three different important criterias: investment rating, hyper score and risk score.

**Are we able to develop a tool that helps users and investors avoiding scam ICOs?**

*Smartract CVA* can help users to avoid fraud smart contracts and create new, secure one. But again, this tool can not give an overall, save rating of an ICO. It should be used as orientation for future investments and decision makings. To get more information about *Smartract CVA* please take a look back at chapter 4.

\(^7\)https://icorating.com/
6. Conclusion

We have analyzed the behavior and vulnerabilities of smart contracts. Also an application to simplify the identification of an initial coin offering has been developed. Chapter 1 gave a short overview of the thesis and the connected research questions.

In chapter 2 we took a small look at related work according to smart contracts, Ethereum and the blockchain in general.

In chapter 3 we have summarized what decentralized systems are. With this background information we were able to explain the base of smart contracts. Due to their Turing complete functionality they provide a huge advantage over Bitcoin. To avoid endless loops, gas (comparable to a kind of currency) is needed for every transaction. A part of the turnover at the beginning of Ethereum is caused by CryptoKitties, a blockchain based cat game. It is one example of what can be achieved with smart contracts in Ethereum. As every technology also smart contracts have their weaknesses. On 18th June 2016 a big attack hit DAO and a hard fork was needed due to the blockchain itself is not immutable. It was caused by using the fallback function. Initial Coin Offerings (ICOs) are also a paradise for criminals. They function as a crowdfunding company. Smart contracts can also include fraud systems like Ponzi Schemes. Section 3.7.4 deals with those problems and shows how you can manually find such systems.

The developed tool Smartract CVA and the included domain-specific language Smartract CVA are explained in detail in chapter 4. Every single step to create and analyze a smart contract / ICO is listed. This tool may be useful for some investors, developers and users. In particular, the user has to go through the whole whitepaper to use step two of the application and fulfills with this reading a main and import task of every ICO rating framework. Chapter 5 evaluates the research itself and the produced tool. The evaluation has been fair but critical and used a scoring method for evaluating the documentation. Our study and tool could be the base of future research in smart contracts analyzing and exploitation.

6.1. Limitations and Future Work

Due to various limitations we were not able to fulfill all desired implementation. For creating a domain specific language we first used MPS, a platform for creating a DSL by JetBrains. It has
many advantages and served well in first position, but we decided to develop a web application and not a client solution. Thus we had to switch the architecture and found TextX, which is similar to Xtext from Java.

To compare our generated contract with the original one we tried Porosity, a decompiler for smart contract. For some contracts the decompiler works well but many of them can not be reproduced/used as expected. After this we came with the idea of comparing the produced transaction of the original contract with fake transaction from our created Solidity contract. This seems as a good solution at first sight, but it has also some limitations. First how should we be able to reproduce the exact same contract as a save contract only with the input of a DSL? This is a really challenging part. One suggestion and try has been with recognizing different patterns like Ponzi schemes in our source code. But this try failed and respectively has not brought the desired results.

Next the DSL is not able to reproduce every smart contract. We are not able to produce objects or too deep nested loops due to the limitations of our Jinja2 template.

We are going to improve our tool and add some useful features to give a more secure rating and guarantee about Initial Coin Offerings. One solution could be a more precious pattern matching between the created and original contract. Another one could be at the transaction or EVM level.

More rules will be added to our developed Smartract DSL and we also want to publish it as a standalone application without the analysis and question part.
Bibliography


A. Appendices

A.1. TAM Questionnaire

Survey for a tool-supported approach to smart contracts

Tool is available for a certain time at the IP stated in your previous mail. If you are not able to use the tool with your browser, please find attached an instruction for installation.

* Required

Part I: General Information

1. Programming Skills *
   Mark only one oval.
   - No skills
   - Basic skills
   - Good skills
   - Excellent skills

2. Blockchain Experience *
   Mark only one oval.
   - No experience
   - Basic experience
   - Good experience
   - Excellent experience

3. Solidity Experience *
   Mark only one oval.
   - No experience
   - Basic experience
   - Good experience
   - Excellent experience

4. Have you ever took part at an ICO crowdsale?
   Mark only one oval.
   - Never
   - No, but I thought about
   - Yes, once
   - Yes, more than once
5. If yes, did you receive any cashout?
*Mark only one oval*
- Yes, higher than my deposit
- Yes, same as my deposit
- Yes, but less than my deposit
- No

**Part II: Smartract CVA**

6. What is the name of the ICO you want to test?

7. On a scale of 1 to 10 - how easy was it for you to complete the survey in step number 1? *
*Mark only one oval*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too difficult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Too easy</td>
</tr>
</tbody>
</table>

8. Did you have any problems with part I of Smartract CVA?

9. On a scale of 1 to 10 - how easy was it for you to recreate a smart contract? *
*Mark only one oval*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too difficult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Too easy</td>
</tr>
</tbody>
</table>

10. Did you have any problems with step II of Smartract CVA?

11. On a scale of 1 to 10 - how easy was it for you to enter the right smart contract address? *
*Mark only one oval*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too difficult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Too easy</td>
</tr>
</tbody>
</table>

12. Did you have any problems with step III of Smartract CVA?
13. How good was your recreation of the original crowdsale smart contract in step II? *
   *Mark only one oval.
   
   Not able to recreate
   
   1  2  3  4  5  6  7  8  9  10  Identical

14. Did Smartrac CVA help you in your decision?
   *Mark only one oval.
   
   ☐ Yes
   ☐ No
   ☐ Other: ______________________

15. What would you improve in Smartrac CVA?
   
   ______________________
   ______________________
   ______________________
   ______________________

Part III: Technology Acceptance

strongly agree = 7
strongly disagree = 1

16. Perceived Usefulness (PU)
   *Mark only one oval per row.
   
   (PU1) Smartrac CVA saves time compared to own detailed research about an ICO
   
   1  2  3  4  5  6  7
   
   (PU2) Smartrac CVA can be used to create smart contracts
   
   1  2  3  4  5  6  7
   
   (PU3) Smartrac CVA can be used to validate smart contracts
   
   1  2  3  4  5  6  7
   
   (PU4) Smartrac CVA can be used to analyze smart contracts
   
   1  2  3  4  5  6  7
   
   (PU5) Smartrac CVA allows me to create smart contracts quickly
   
   1  2  3  4  5  6  7
17. Perceived Ease of Use (PEU)

Mark only one oval per row.

- (PEU1) Using Smartract CVA is easy for me
- (PEU2) Complete step one of Smartract CVA is easy for me
- (PEU3) Creating a smart contract (step two) with Smartract CVA is easy for me
- (PEU4) Creating a smart contract (step two) without an offered template is easy for me in Smartract CVA
- (PEU5) I do not have any problems buildings a DDL in Smartract CVA on my own (without Samples)

18. Behavioural Intention to Use (BI)

Mark only one oval per row.

- (BI1) I will try to use Smartract CVA instead of any other research method about an ICO
- (BI2) I will create smart contracts with Smartract CVA
- (BI3) I will analyze ICOs with Smartract CVA

19. Attitude Toward Using (ATU)

Mark only one oval per row.

- (ATU1) I prefer using Smartract CVA over any other related tool
- (ATU2) I prefer using Smartract CVA contract validator over another related tool
- (ATU3) I prefer using Smartract CVA contract creator over other related tools
- (ATU4) I prefer using Smartract CVA contract analyzer over other related tools