IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

An Ethereum based Crowdfunding Platform for Open Source Software Development

Author:
Timothy Lim

Supervisor:
Dr. Michael Huth

Submitted in partial fulfillment of the requirements for the MSc degree in Msc. Computing of Imperial College London

September 2016

Abstract

Open source software has come to represent a major proportion of all software in production. The advantage of open source software being free has not only resulted in its widespread adoption but also a new form of collaborative programming in which all users of the software can both contribute and benefit collectively. However, the problem of long-standing unresolved bugs in large open source projects pose a danger to companies reliant on open source software.

There have been some attempts at creating platforms to enable the public to collectively fund the resolution of issues in open source repositories but their success has not been comprehensive in part due to the high cost of maintaining infrastructure that can efficiently handle the collection and payment of rewards.

The motivation of this project was to leverage new computing paradigm provided by the Ethereum blockchain to investigate how a crowdfunding platform for open source issues could be made more efficient. Based on the Bitcoin blockchain, Ethereum provides limited cloud-computing environment by allowing complex scripts to be run and computed by anonymous nodes in the network.

The main achievement of the project was to demonstrate that a basic bug bounty scheme for open source software development could be created and deployed on the Ethereum network. The first stage of the project was to capture the user stories associated with existing platforms, translate them into a prototype whose functionality was rapidly iterated on a local simulated blockchain. At the conclusion of the project, the prototype was successfully deployed onto the Ethereum Testnet and the smart contracts that comprised the logic of the program was written onto the global Ethereum blockchain.

Acknowledgments

I would like to thank my supervisor, professor Michael Huth for his considerate and valuable support throughout this project. I thoroughly enjoyed the conversations we had during the summer and appreciate his feedback on the project.

I would also like to thank professor William Knottenbelt for introducing me to Ethereum and for his infectious enthusiasm and compulsion for speaking the truth to the power.

Finally, this project would not have been possible without the kind help of Raine Revere whom I consulted extensively during the development of the software.

Contents

Chapter 1

Introduction

1.1 Introduction

“Software is eating the world”, claimed entrepreneur and investor Marc Andreessen in 2011. Writing in an op-ed published by the Wall Street Journal, he describes the current state of the world as one in which

> “More and more major businesses and industries are being run on software and delivered as online services—from movies to agriculture to national defense. Many of the winners are Silicon Valley-style entrepreneurial technology companies that are invading and overturning established industry structures” [14]

The fundamental point that Andreessen is making is that technological innovations have the potential to change the processes that run our day to day lives. More specifically, inventions in software have the lowest barrier to entry as the only cost to produce new software are developer hours. Therefore, the largest gains tend to come from software and the companies with the greatest amounts of growth will also tend to be software companies.

Within this wave of technological productivity is a less visible undercurrent defining the types of technology that has become popular amongst developers and companies. Traditionally, software has been created and sold much like any other type of intellectual property. Software is written and then each new copy is licenced out to users in the form of separate products. The Open Source movement sought to make the source code of all software free and open to modification and further distribution. The idea motivating open source programmers was that if one were to own a copy of a program he or she should have the right to modify and customize it as they saw fit [25].

Open Source software has come to play an integral part in the software industry. The open source operating system Linux “powers more than 95 percent of the top 1 million domains.” In addition, ”most of the world’s financial markets run it, and so do 98 percent of the top 500 fastest supercomputers. More than 75 percent of cloud-enabled enterprises say it’s their primary cloud platform” [23]. Immediately one can see that since open source software is free, it also means that companies cannot charge for it. The details of open source business models, and their associated problems, are explored later in the next chapter.

The key contribution that this project is a platform that allows the public to fund additional features for open source software. The idea is that since open sourced software is open and freely available to the public, it follows then that there should be a method by which the public can propose and crowdfund the development of new features and bug fixes.

The second contribution of this project is the choice of technology the crowdfunding platform will be built on. Released in July 2015, Ethereum is a protocol that allows computers in a network to co-operate and collectively act as a virtual computer. Currently, web applications are programs that are constantly run on proprietary servers run by specific companies. Instead of using a remote computer from a single cloud computing provider (e.g. Amazon Web Services) it is possible to use the Ethereum Virtual Machine. This project aims to deploy an application on the Ethereum Virtual Machine whose computation is parceled out throughout the Ethereum network guaranteeing application up-time as long as the network is active.

The objectives of this project are therefore to:

1. Introduce the concept of blockchain-based networks and a high level understanding of Ethereum
2. Analyse and capture the key user stories from existing crowdfunding platforms and rework them into an Ethereum context
3. Develop a prototype application: Encode the logic of a crowdfunding platform into a proof of concept that can run on a local simulation of the Ethereum Network
4. Deploy the prototype to the Ethereum Network: Ensure the basic functionality can be computed and verified by external anonymous nodes and that the application is ’live’
5. Evaluate the feasibility of the prototype and examine its effectiveness compared to existing solutions

1.2 Report Outline

A brief introduction and summary of each chapter:

Background

- Ethereum is introduced in relation to Bitcoin and blockchain technology. In addition, the origins and problems surrounding open source software development are presented.
- Bountysource, a crowdfunding platform for developers, is explored and its functionality is documented as material to shape the requirements of the project.

Design

- A prototype architecture of the system is proposed and discussed in relation to requirements.
- Ethereum smart contract best practices are discussed and potential security flaws are noted.

Prototype Implementation

- The first iterations of the platform are documented as well as the tools and frameworks that were experimented with and used during development.

Deployment To The Ethereum Network

- The Ethereum Main Network and Test Network are introduced and their differences elaborated. A deployment procedure is outlined and details of writing the code to the ethereum blockchain are documented.

Evaluation

- The objectives stipulated in the introductory chapter are reviewed and the work done on the project thus far is evaluated in relation to them.
- Two major bugs are noted in a subsection entitled Future Work as well as possible extensions to the project.

Chapter 2

Background

2.1 Blockchains

2.1.1 Bitcoin

Bitcoin is the world's first completely decentralized digital currency [19]. Bitcoin was revolutionary because it solved the key challenge that faces all types of digital tokens that seek to act as stores of value: the problem of double spending. In a digital realm, duplication is trivial. Due to the fact that without a central authority, such as a bank or a centralized payments system like Paypal, there is no way to prevent holders of a particular token from double spending a digital token.

The inventor of Bitcoin, Satoshi Nakamoto, resolved the 'double spend' problem by proposing a system of open consensus. All nodes operating in the Bitcoin network carry a copy of a global Bitcoin ledger. On this ledger all accounts in existence and their respective balances are made public. Each node attempts to update the global ledger by solving a cryptographic puzzle as specified by the Bitcoin protocol. As soon as a node has solved the puzzle, it is given the privilege to update the global ledger with the transactions that have occurred after the last update. All other nodes check that the winning node had indeed solved the puzzle and if it has, they propagate the latest version of the global ledger. Although this may seem incredibly inefficient, it has solved the problem of double spending by forcing a consensus amongst all nodes in the network not through trusting an authority but by simply following a set protocol. This distributed global ledger called the blockchain as each new 'step' added by the winning node is called a block and the end result of the transaction history of Bitcoin is a chain of blocks [7].

In the case of Bitcoin, these blocks of data represent the balances of all the accounts of Bitcoin wallets at a particular point in time. Transactions of Bitcoins from one account to another are processed in a single batch and the creation of a new block to be added to the blockchain with the new balances of the accounts whose transactions have been computed by the network represents a 'step' in the timeline of the Bitcoin ecosystem.

2.1.2 Introduction to Ethereum

Ethereum is an internet service platform for guaranteed computation that builds upon the innovation of Bitcoin's blockchain [2]. Instead of utilizing a blockchain as a simple store of accounts (i.e. a ledger), Ethereum uses it as a state of a virtual computer. Ethereum does this by building what is essentially the ultimate abstract foundational layer: a blockchain with a built-in Turing-complete programming language. This allows anyone to write scripts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions [2].

Superficially, Ethereum may appear to be an extremely cumbersome and limited version of cloud computing. In fact, Ethereum may be compared to mainframes of the past as it 'notionally has only has a single processor (no multi-threading or parallel execution) but as much memory as required' [2]. However, the trade-off is made for the ability to run applications whose code is openly visible and auditable and run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference [2]. In this way, the Ethereum Virtual Machine is in a sense a global computer whose latest state in memory is represented by the latest block in the Ethereum Blockchain.

The Proof of Work model employed by the Ethereum Network is a derivative of Bitcoin's. Just as in the Bitcoin network, consensus in the Ethereum network is achieved by the discovery of the block with the highest difficulty which all other nodes on the system check for validity [2]. Ethereum uses a Proof of Work algorithm called Ethash that requires miners to find a nonce such that the a level of difficulty is maintained [2]. This PoW model is a slight departure from Bitcoin's as the difficulty is adjusted to produced a new block every 12 seconds, as opposed to 10 minutes. The token created by the mining of blocks in the Ethereum econosystem is called ether.

2.1.3 Scripting On A Blockchain

Smart contracts are pieces of code that exist on the Ethereum Blockchain. Ethereum improves upon Bitcoin's basic scripting functionality by allowing applications to be written by various pseudo-turing complete programming languages. Smart contracts can read other contracts, make decisions, send ether and execute other contracts. The combination of several smart contracts working in unison leads to the possibility of building complex distributed applications that are transparent in their execution and censorship proof (as long as the Ethereum network operates).

It is important to note that "contracts" in Ethereum should not be seen as something that should be "fulfilled" or "complied with"; rather, they are more like "autonomous agents" that live inside of the Ethereum execution environment, always executing a specific piece of code when "poked" by a message or transaction, and having direct control over their own ether balance and their own key/value store to keep track of persistent variables [16].

The strength of the Bitcoin network is that as long as nodes continued to mine and process transactions, it was impossible for any third-party to prevent the exchange of Bitcoins between wallets. Ethereum leverages the same advantage to create a virtual machine which will execute code as long as the nodes in its networks continued to support it. The decentralized nature of both of these systems means that the networks are impervious to denial of service attacks since any slack in the network causes the rewards to increase, incentivizing additional nodes to come online. Just Bitcoin may be considered an 'uncensorable' digital currency, the Ethereum network could in turn be considered an 'uncensorable' virtual computer.

2.1.4 Ethereum State Transitions

In Ethereum, the state is made up of objects called 'accounts', with each account having a 20-byte address and state transitions being direct transfers of value and information

between accounts.
Each account in Ethereum contains four fields [16]:

1. The nonce, a counter used to make sure each transaction can only be processed once
2. The account's current ether balance
3. The account's contract code, if present
4. The account's storage (empty by default)

Transaction fees are paid for by ether which is, as explained above, produced by the mining of new blocks.
There are two types of accounts:

1. Externally owned accounts, controlled by private keys. It may be instructive to view these accounts as representations of avatars of entities that exist outside the network. Externally owned accounts do not hold any code but are able to send messages to contract accounts thereby activating them.
2. Contract accounts, controlled by their contract code. Contract accounts are essentially in-state wrappers for contract code. They exist to hold code on the blockchain that can be activated by receiving a message from an externally owned account or by another contract account. Once it has been activated, it can read and write to internal storage within its own account or even go on to message other contract accounts.

2.1.5 Distinguishing Between Messages and Transactions

The terms 'transaction' and 'message' have been used fairly loosely and it is worth discussing the difference between the two concepts. The term "transaction" is used in Ethereum to refer to the signed data package that stores a message to be sent from an externally owned account [16]. The first three components of a transaction in ethereum are similar to that of any other cyrpocurrency system. The final two components, however, represent a crucial difference that distinguishes ethereum as a blockchain-based computational platform:

1. The recipient of the message
2. A signature identifying the sender
3. The amount of ether to transfer from the sender to the recipient
4. An optional data field
5. A STARTGAS value, representing the maximum number of computational steps the transaction execution is allowed to take

6. A GASPRICE value, representing the fee the sender pays per computational step

The Ethereum network places a price on computation to prevent "accidental or hostile" infinite loops and other types of runaway programs [16]. The STARTGAS and GASPRICE fields make the network robust to denial-of-service attacks as programs are inherently limited in the number of computational steps that they are allowed to execute.

Gas is simply Ethereum's way of representing ether to pay for transaction fees. Currently the price of gas is fixed to 1/100,000 of an ether and as Bitcoin miners prioritise transactions willing to pay the highest fee (in Bitcoin) so do miners on the Ethereum network.

Messages are similar to transactions in that they are data sent between accounts but messages are virtual objects that are sent only from contracts to other contracts whereas transactions are data packages sent only from external accounts. Transactions are therefore representations of interactions that entities external to the blockchain use to exert whereas messages are the internal form of communications between components of the system.

2.1.6 Solidity

According to the official Ethereum documentation:

> "Solidity is roughly speaking, an object-oriented language designed for writing contracts in Ethereum. Contracts are (typically) small programs which govern the behaviour of accounts within the Ethereum state. These programs operate within the context of the Ethereum environment. Such accounts are able to pass messages between themselves as well as doing practically Turing complete computation." [**?**]

Solidity is similar in its syntax to Javascript and other C-like languages but has been designed to accommodate calls to the Ethereum blockchain with special types such as $address$ (a 160 bit Ethereum account identifier) and keywords such as $contract$ (which refers to a contract object).

2.2 Open Source Software

2.2.1 A Brief Introduction to Open Source Software

Open Source software is a model for software distribution. Typically, software is only sold as a finished product and the user is often only allowed a limited amount of installations. If the user would like to exceed the number of installations, she would have to often purchase a new copy of the software. The firm that created the product has total control over it; if there is a bug, users have to wait for the firm to fix it and if the firm goes out of business or discontinues support of a product, users of that product have no recourse [15].

By contrast, open source software places the control of the software into the hands of its users. The software itself is licensed to guarantee free access to the source code. This allows the user to install the software on a new platform without an additional purchase, to get support (or create a support mechanism) for a product whose creator no longer supports it [15].

Through sharing the source code of their products, open source software firms imply a different understanding of their users than that of a traditional firm. Open sourced software makes its digital blueprints fully accessible under the assumption that some of its users would not only be willing and able to fix bugs in the code but to even contribute to improve it. This understanding of the user as a potential developer effectively grants ownership and stewardship of the tool to the person that uses it.

It was in this seemingly anarchic delegation that almost all work and "being open to the point of promiscuity" [15] that lay at the foundation of the Open Source model of software development being an innovative process. In contrast to the 'cathedral builders' of Microsoft and other traditional software companies, open sourced software was designed to be a collaborative creative effort from all the users that were willing to contribute.

The idea of open source software has its roots in the free software movement of the 1980's. Richard Stallman, who was working at MIT, launched the GNU project with the aim of developing a UNIX-like operating system to be distributed for 'free'. This notion of 'freedom' is clarified by Stallman as the following:

> "Free software" means software that respects users' freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software. Thus, "free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer". We sometimes call it "libre software" to show we do not mean it is gratis. [25]

The notion of open source software as we know it today, is more or less form of branding placed on Stallman's 'free software':

> "The Open Source Initiative is a marketing program for free software. It's a pitch for "free software" on solid pragmatic grounds rather than ideological tub-thumping. The winning substance has not changed, the losing attitude and symbolism have." [18]

Both ideologies enshrine the ability for users to have access and control of the source code of the program in order to improve to suit his or her need. The difference comes in the licencing of derivative copies of the software where the Open Source movement has opted to allow derivatives to be ’closed source’ whereas ’free software’ can never be propriety. Hence the creation of more permissive licences such as the MIT and BSD licences.

For the purposes of this project, we will focus on the innovation of collaborative creation that is espoused by both the free software and the open source movements. While the nuances of licencing and the ’freedom’ of derivative works are critical issues in the long run, it is valuable to examine how micro-payments can be brought to bear on the already very successful collaborative model of work.

A leading example of the open source method of collaboration has been the incredibly fast and secure development of the Linux kernel. Since 1991, there have been a new version of the operating system released approximately every week [26]. Since the public backing and investment of Linux by IBM, the operating system has gone from strength to strength. Today, Linux powers 79.3 percent of all smartphones, 85 percent of world’s equity and trading platforms and exists as the operating system for 92 percent of the world’s high performance computing systems (The Linux Foundation).

2.2.2 Models of Funding for Open Source Software

Bill Gates prophesied the financial difficults that would haunt future open source projects. Writing in an open letter to early computer hobbyists and tinkerers in 1976, he critised the sharing of Microsoft’s code and likened it to theft:

> Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software... Most directly, the thing you do is theft. [17]

Given the success of Linux, Gate’s pessimistic view of Open Source appears to be presumptuous in hindsight. However a brief systematic look at the models of funding for Open Source projects may elucide some of his concerns. In a paper commissioned European Union, Brock et al. list the main business models of Open Source companies:

1. Twin licensing (same as Dual licensing): for companies offering the same software code distributed under GPL and a commercial license;
2. Split OSS/commercial products (same as Open Core): distinguishing between a basic open source software and a commercial version, based on the community
3. Badgeware: for companies distributing software under MPL with the addition of a “visibility constraint”, i.e. the compulsory inclusion of visible trademarks, logos, and copyrights notice in the user interface;
4. Product specialists: companies that create or maintain a specific software project and use a “restrictive” open source license to distribute it;
5. Platform providers: for companies providing selection, support, integration, and services on a set of projects, collectively forming a tested and verified platform;

6. Training and documentation: companies that offer courses, online and physical training, additional documentation or manuals. This is usually offered as part of a support contract, but recently several large scale training center networks started [13]

As shown above, the funding of actual development work is often fairly indirect and companies have to rely on ancillary services to generate a profit. In this sense, the business of Open Source is often orthoganal to the writing of the code whereas companies that sell propriety software, the sales from the product are directly linked to its development.

Essentially, one of the most common methods of open source 'funding' is the sponsorship of software by large companies who seek to profit through the positive side-effects that come with widespread adoption of the said software. For example, Google moved to displace Apple's iOS marketshare by releasing its Andriod operating system under the Apache 2.0 Lisense which allowed it gain marketshare from a mere 2 percent in 2009 to over 80 percent in 2013.

While the sponsorship of projects such as Andriod, Swift, Java (by Google, Apple and Oracle respectively) has resulted in successful worldwide adoption of their products and services, it does not provide a sustainable business model for projects without a parent company.

For the larger Open Source projects, non-profit organizations are formed to act as caretakers in the form of foundations. Open Source foundations benefit from the following attributes:

1. Ability to raise funds more easily/effectively
2. They can give donors a tax benefit, and avoiding taxes on funds raised
3. It may help them create a governance structure
4. It gives greater brand recognition
5. There's too much money and associated accounting/bookkeeping/filings for one person to handle
6. They are worried about contributor liability, e.g. if organising a conference
7. It can hold assets (e.g. trademarks or domain names) on behalf of the project[12].

On occasion, Open Source foundations appeal to its userbase directly. Two alternative funding models are of particular interest to this project: voluntary donations and bounty driven development. The Mozilla Foundation supplements its income by hosting annual fundraisers, raising 12.6 million USD in 2014 from 380 000 different individuals.

The Mozilla Foundation then in turn reinvests a portion of that funding to sponsor its Bug Bounty Program. The Mozilla Bug Bouty program is designed to encourage security research in Mozilla software and to rewared those who find the bounty for valid potentially exploitable critical and high security rated client security vulnerabilities. Rewards are often between 3000 and 7500 (USD). The bounty program encourages the earliest possible reporting of these potentially exploitable bugs (The Mozilla Foundation).

2.2.3 Funding Failures in Open Source Software

In April 2014 a bug called 'Heartbleed' in the open source OpenSSL cryptography library was disclosed to the public. At the time of disclosure the bug affected 17% of SSL web servers which used certificates issued by trusted certificate authorities [10]. The combined market share of just those two out of the active sites on the Internet was over 66% according to Netcraft's April 2014 Web Server Survey [11]. Even though OpenSSL had its own foundation it only received "US$2000 a year in outright donations and sells commercial software support contracts" [22]. Steve Marquess, one of the co-authors of OpenSSL, notes that "in the five years since it was created OSF has never taken in over $1 million in gross revenues annually." [22].

The Heartbleed bug showed that if an open source project falls outside the proven business models detailed in the previous section, they can struggle to cover their costs. The fact that 66% of the web including "Fortune 1000 companies ... the ones who [included] OpenSSL in firewall/appliance/cloud/financial/security products that [were sold for a] profit" came to abuse an almost bankrupt foundation with a single employee proves that solutions need to funding developer hours need to be found [22].

In addition to the risk of hitherto undiscovered bugs lurking in widely used open source libraries, there exists an equally dangerous risk of reported bugs that have not been fixed. Kim and Whitehead of U.C. Santa Cruz mined the software histories of ArgoUML and PostgreSQL (both large and widespread open source libraries) to determine the nature of the bugs in the codebase and the length of time it took to solve them. They conclude the following:

> We computed and analyzed the bug-fix time of each file. We believe that bug-fix time is useful, and should be widely used for bug related analysis [21].

Saha et. al's paper entitled "An Empirical Study of Long Lived Bugs" found that " More than 90% of these long lived bugs affected users' normal working experience. The average bug assignment time was more than one year and the bug fix time after the assignment was another year on average."
[24]
Most importantly, they conclude that:

> While problem complexity, problems in reproducing, and not understanding the importance of some of the long lived bugs in advance are the common

> reasons, we observed there are many bugs that were delayed without any specific reasons. Finally, by investigating the actual source code changes for these long lived bugs, we noted that a bug surviving for a year or more does not necessarily mean that it requires a large fix. In fact, we found 40fixes that involved few changes in only one file [24].

This result is one of the key motivators for this project. With these findings in mind, it is the goal of the project to place a monetary value on bugs in open source libraries to engineer fixes that may not happen otherwise.

2.3 Bountysource

Bountysource (www.bountysource.com) is a web application that is one the main inspirations for this project. It is a marketplace for maintainers and users of open source projects to post bounties for bugs and feature requests. It has been in operation since 2012 and in October 2015 posted a record amount $80 000 USD worth of bounties [1]. As a proof-of-concept, Bountysource is extremely valuable as it shows that there is a market for modular development of open source projects in addition a strong acceptance of the crowdfunding model.

At its core, the main function of Bountysource is to act as an escrow holding a bounty and paying out when given the permission. It acts as a trusted third party between people who want code to be written and developers in addition to being a central place for issues with bounties to be found. The simplicity of the role of Bountysource lends itself well to Ethereum smart contracts as the main strength of the Ethereum system is to encode the logic of trusted third parties into open and auditable scripts. A theoretical implementation of Bountysource will be less of a platform but more of a protocol (whose logic is encapsulated in smart contracts) for developers to interact with users and maintainers of projects.

There are three potential advantages of an Ethereum-based implementation of Bountysource:

1. Security: Both users and developers have to trust that Bountysource will store their funds securely. If bountysource.com is hacked, there is limited recourse to recover lost funds. In addition, as Bountysource grows in size it becomes increasingly attractive for hackers to attempt to drain their funds. However, if funds are stored in individual smart contracts there is no one 'honey pot' for hackers to attack. Furthermore, the code for each smart contract can be made so simple as to reduce any room for malicious use. Since each smart contract deployed to the network is available for auditing, users do not need to trust a platform. Instead, they can read the smart contract for themselves.

2. Transaction Fees: In the Ehtereum network, ether is transferred between accounts much like Bitcoin is between wallets in the Bitcoin network. Transferring ether costs roughly 0.002% of the value being transferred. Bountysource, on the other

hand, charges a 10% withdrawal fee. It is hoped that having a negligible transaction fees will allow micro-crowdfunds to occur, perhaps to as little as $2 U.S. dollars for an issue. Such small bounties are only possible on a blockchain-based platform and has never been attempted before.

3. Autonomy All coded deployed on the Ethereum blockchain is inherently open and visible for anyone to see. As such, anyone can write and deploy their own version of the project. There is no obligation to be on a certain platform and each open source project can fork and implement a different version of the smart contracts.

2.4 Open Source Workflows

As the aim of this project is to find a workable solution to create a market for unresolved bugs in open source software as well as underfunded feature requests. Therefore, it is important that a discussion on the software development life cycle of open source projects is included in the design. By examining the process by which code is written for open source projects, we can target where a crowdfunded solution would be appropriate and design to those requirements.

2.4.1 Software Development Lifecycles

A traditional software development life cycle is presented by Saini and Kaur below:

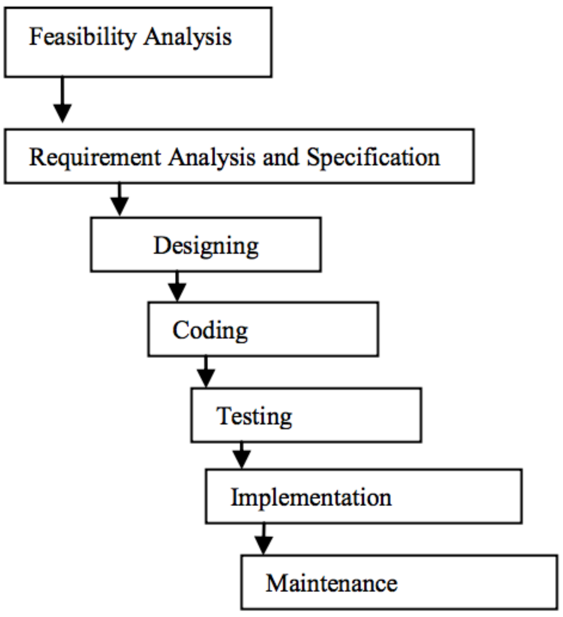


Figure 2.1: A Traditional Software Development Lifecycle

This is also referred as a 'waterfall' model of development as each step in the process naturally leads to the next in a permanent and non-iterative fashion.

Open Source Software projects, by contrast, rely on a more iterative ’agile’ development methodology. Furthermore, a ’waterfall’ model assumes that all processes are within the control of a single firm (either directly through employed teams or indirectly through contract workers). Open source projects, on the other hand are by nature reliant on informal work done by people not employed by the company and thus this dimension has to be reflected in modelling workflows in such projects.
Jørgensen elaborates on the iterative nature of open source development:

> Coding and pre-commit testing takes place in the committer’s private repository. Development release of a change is the integration of it into the repository’s development branch where it resides during parallel debugging... Significantly, the project [BSD] does not have guidelines for allocating people to work on a change in the first place. The project maintains a prioritized list of tasks, but there is no obligation on behalf of a maintainer to pick up a task inside his area of responsibility [20].

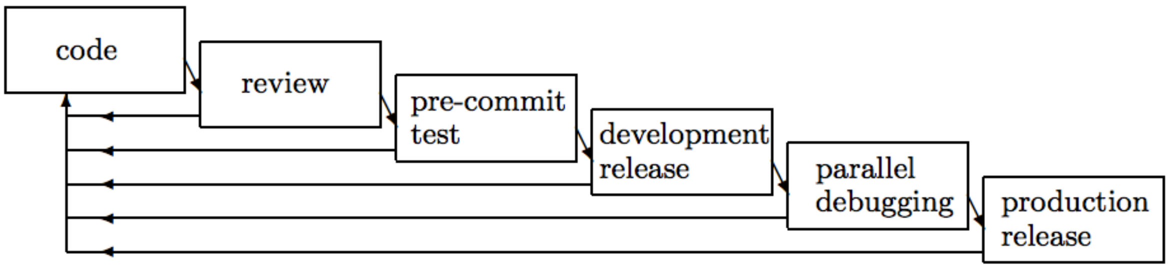


Figure 2.2: Lifecycle of Changes in an OSS Project

Jørgensen notes that the nature of collaboration is decentralized and often asynchronous with bugs and features being fixed and updated in private repositories and commonly not to any particular development plan. The focus of this project should therefore be on helping to expose bugs that have been uncovered as the main development branch has moved ahead and incentivize their resolution even as development by the core team progresses.

With regard to the design and implementation of new features, it is highly variable on the nature of the project and the internal guidelines of the core development team. While bug fixes are fairly easy to evaluate as the description of the bug is itself a miniature software requirement and it is implicitly assumed that all stakeholders want a bug to be fixed, for features it is not clear how the requirements would be gathered and if a core team would even be receptive of an unplanned piece of code being part of their repository.

2.4.2 Git

Git is a version control system used for software development. Git encourages users to organize their work in the form of ‘branches’. When a project is initialized all the files are tracked on the main branch, often called the master branch. The master branch is often reserved for a stable version of a project that has passed all build tests. In order to develop and test new features without affecting the stable version of the project, Git allows users to fork a copy of the main branch and work with a duplicate copy of the code. When a feature has been successfully tested it is then merged back to the main branch.

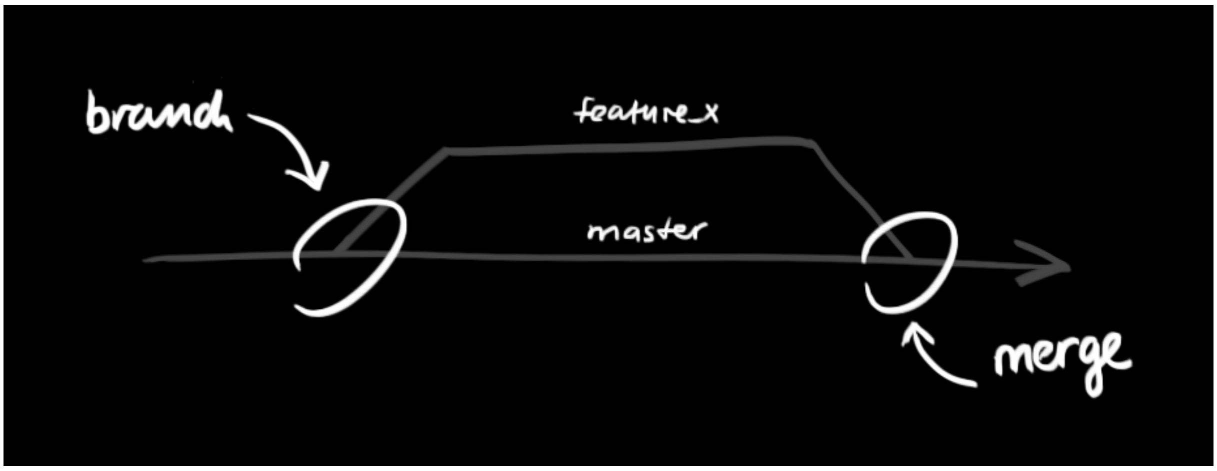


Figure 2.3: Forking and Merging in Git [?]

2.5 Summary

Open source software has achieved widespread adoption but many public and extensively used projects have outstanding bugs (both reported and not). This poses serious security threats for all projects that depend on the open source libraries, one of the latest and most devasting being the Heartbleed bug. Bountysource is crowdfunding platform that allows open source organizations to post bounties for bugs found in their codebases and has proven to be a moderately successful solution.

Ethereum is a novel technology based on work done on Bitcoin that provides a virtual machine whose computations are executed by nodes in the network. It can be used as both a network to transfer unreproducible tokens as well as a programming environment to develop complex applications. This project proposes using Ethereum to build a proof of concept to improve the efficiency of Bountysource.

Chapter 3

Design

3.1 Architecture

3.1.1 Requirements

At its most fundamental level, the purpose of this project is to incentivize the writing of open source code by placing a monetary (denominated in a crypto-currency or otherwise) value on a proposal. Broadly speaking, there are three parties involved in every crowdfund:

1. The Public:

   The general public with access to the platform can create a bounty for a bug that they have found or for a feature that they would like to see be built and integrated into the open source project.

2. The Developers:

   Developers can see which bounties are available and choose to write code to satisfy requirements as detailed by the public. As in traditional git fashion, they will fork a branch, develop their own solution and submit a pull request to the owners of the open source repository.

3. The Repository Owners/Maintainers:

   The maintainers are essentially the arbiters of the crowdfunds. If they deem a pull request to be of a sufficient standard to be merged to a branch in production (and it's merged) it is their responsibility to mark the crowdfund as successful in order to release the funds to the developer.

Taking inspiration from bountysource.com, we can breakdown the fundamental logic of a crowdfunding platform into the following three outcomes:

1. A proposal is successfully funded:

   The owner of the repository merges the pull request into the main branch, or any other branch in production and the developer is paid the bounty. The code is crowdfunded by the public and by being merged to repository, is available to be used by the public.

2. A proposal fails to be funded:

   If a proposal cannot raise enough money to fund a solution it should be withdrawn and all the money should be returned to the funders. In order to not have funds sitting in a smart contract indefinitely it makes sense to have a timeout feature (either a hard deadline of 30 days or perhaps set by the funders).

3. A proposal is successfully funded but is not approved by the maintainers of the project:

   Even if the code is written, there is no guarantee that the maintainers of a project are going to accept it. This could be due to security reasons or to prevent bloating the codebase. There are two ways to resolve this problem, either each proposal has to be vetted by the repository owners before funding begins or the platform can give the repository owners the ability to cancel any crowdfund at anytime. The latter option seems more suitable for a basic prototype.

The relationships between the three stakeholders is represented in Figure 3.1. As shown in the diagram, a smart contract sits in between the stakeholders as it encapsulates all the logic to execute the outcomes listed above.

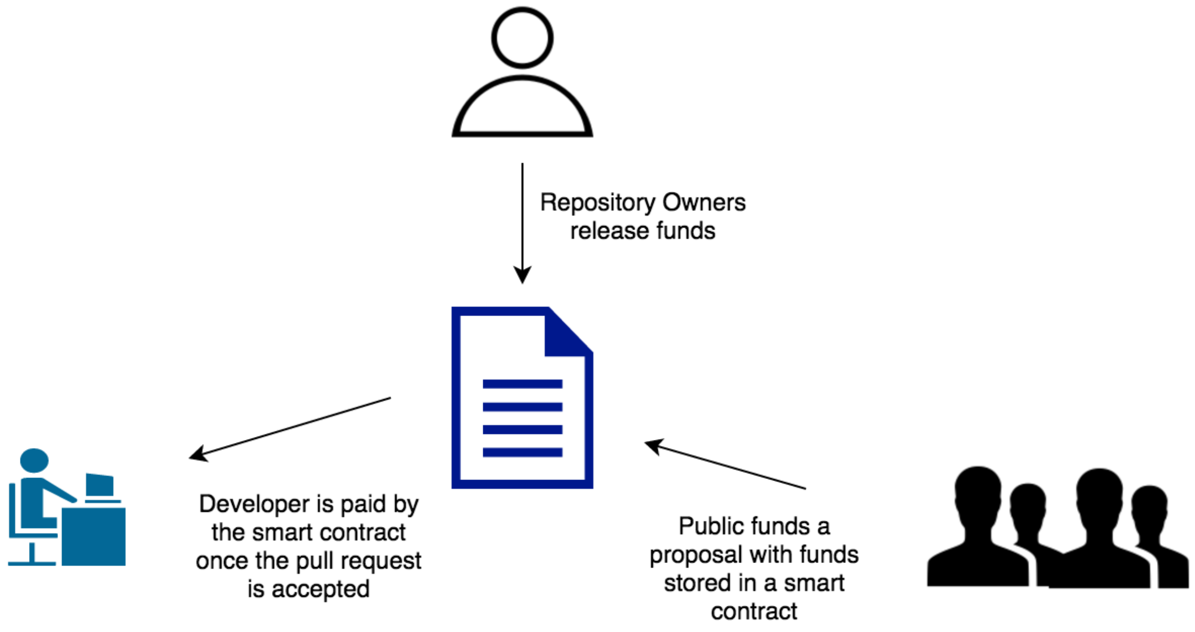

Repository Owners release funds
Developer is paid by the smart contract once the pull request is accepted
Public funds a proposal with funds stored in a smart contract

Figure 3.1: Stakeholder Relationship

3.1.2 Smart Contract Design

The three users stories for the a crowdfuded proposal can be encapsulated in an Ethereum smart contract. Let us simply refer to this smart contract as ‘Crowdfund’. Since we have chosen the programming language Solidity, we are endowed with its built in object oriented capabilities. This means that for each proposal initiated by the public all that is needed is to instantiate a new instance of a ‘Crowdfund’ class. We summarize the general capabilities of such a class in Figure 4.

The Crowdfund smart contract class contains information necessary for its internal logic as well as meta-data that it can return if queried by other contracts. The two key functions of a Crowdfund are sendTo(address) and cancel(). Sendto(address) is called when the Crowdfund has been successfully funded and this sends the total funds to the address specified in the parameter. Cancel() simply cancels the fund and reverts all the funds back to the funders as the amounts that each funder contributed are stored in the

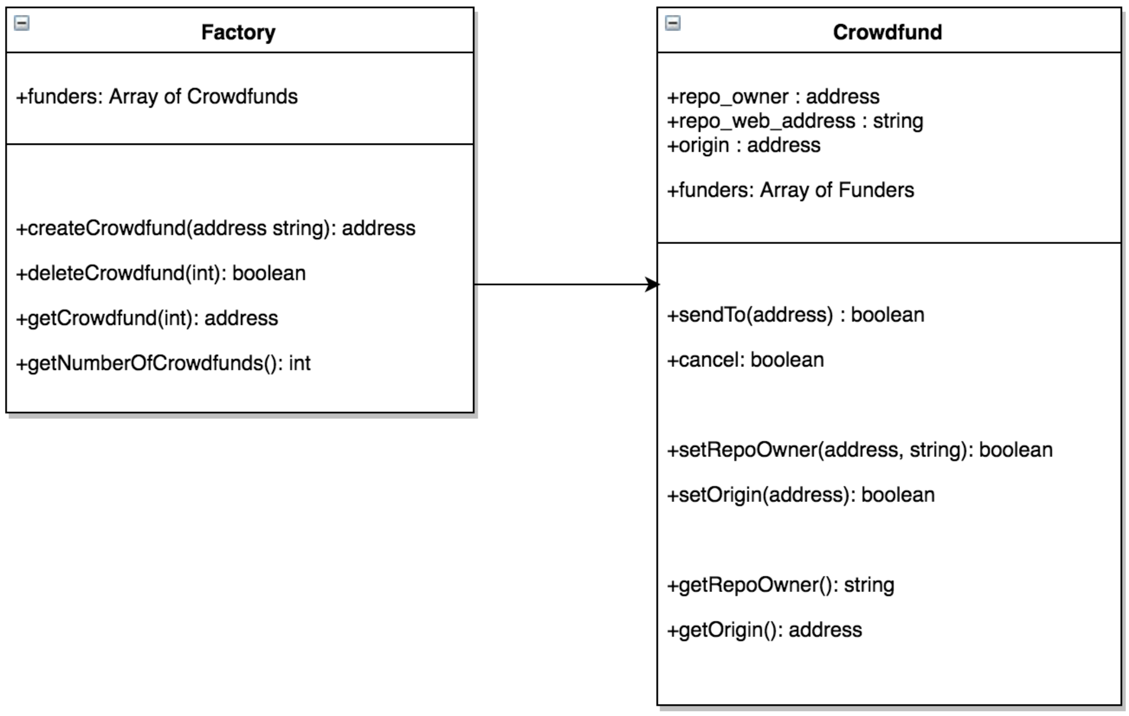

Factory
+funders: Array of Crowdfunds
+createCrowdfund(address string): address
+deleteCrowdfund(int): boolean
+getCrowdfund(int): address
+getNumberOfCrowdfunds(): int
Crowdfund
+repo_owner : address
+repo_web_address : string
+origin : address
+funders: Array of Funders
+sendTo(address) : boolean
+cancel: boolean
+setRepoOwner(address, string): boolean
+setOrigin(address): boolean
+getRepoOwner(): string
+getOrigin(): address

Figure 3.2: Crowdfund Class

funders array.

Each Crowdfund should also contain meta-data about itself for additional security purposes. For a prototype stage, we can include a string representing where on the Internet potential funders and developers can find a cross reference of this proposal. Each crowdfund also has axillary functions in the form of 'getters' that facilitate the returning of data specific to this Crowdfund. Although not visible in Figure 4, each smart contract written in solidity contains a default function that handles the logic for when funds are sent to its address. In our case, we just add a new funder to the array of funders which includes data on how much that funder contributed and from which address the funds were sent from.

However, a platform needs to host more than a single Crowdfund. The purpose of the project is to create a way to instantiate multiple Crowdfunds depending on the specifics of each new software proposal. Therefore, a Factory class to create Crowdfunds is necessary. As shown in Figure 4, a Factory creates new Crowdfunds taking in the details necessary such as the address of the repository owner and the meta-data of where a cross-reference of the propsoal exists online. A secondary function of the Factory contract is to act as a directory for all the Crowdfunds that it has created and are currently active i.e. all the active proposals on the platform. Other contracts and views residing on the Internet can mine data (and data from specific contracts) from this directory by accessing its 'getter' functions just as with the Crowdfunds themselves.

3.1.3 Web 3.0 Architecture

Applications built on the Ethereum network are referred to as Distributed Applications or DApps for short. The structure of a typical Web 2.0 web application consists of a model, view and controller (MVC). The model refers to the model of the data i.e. the database, the view is the presentation layer that users interact with and the controller is the logic that bind the two. The model is hosted on a server and part of the compiled code is served to the view (depending on the nature of the web app).

A Web 3.0 application built on Ethereum mirrors the structure of a Web 2.0 quite closely save for the fact that the model resides on the Ethereum blockchain and the controller that binds the view to the blockchain has to be precompiled in an Ethereum based language such as solidity.

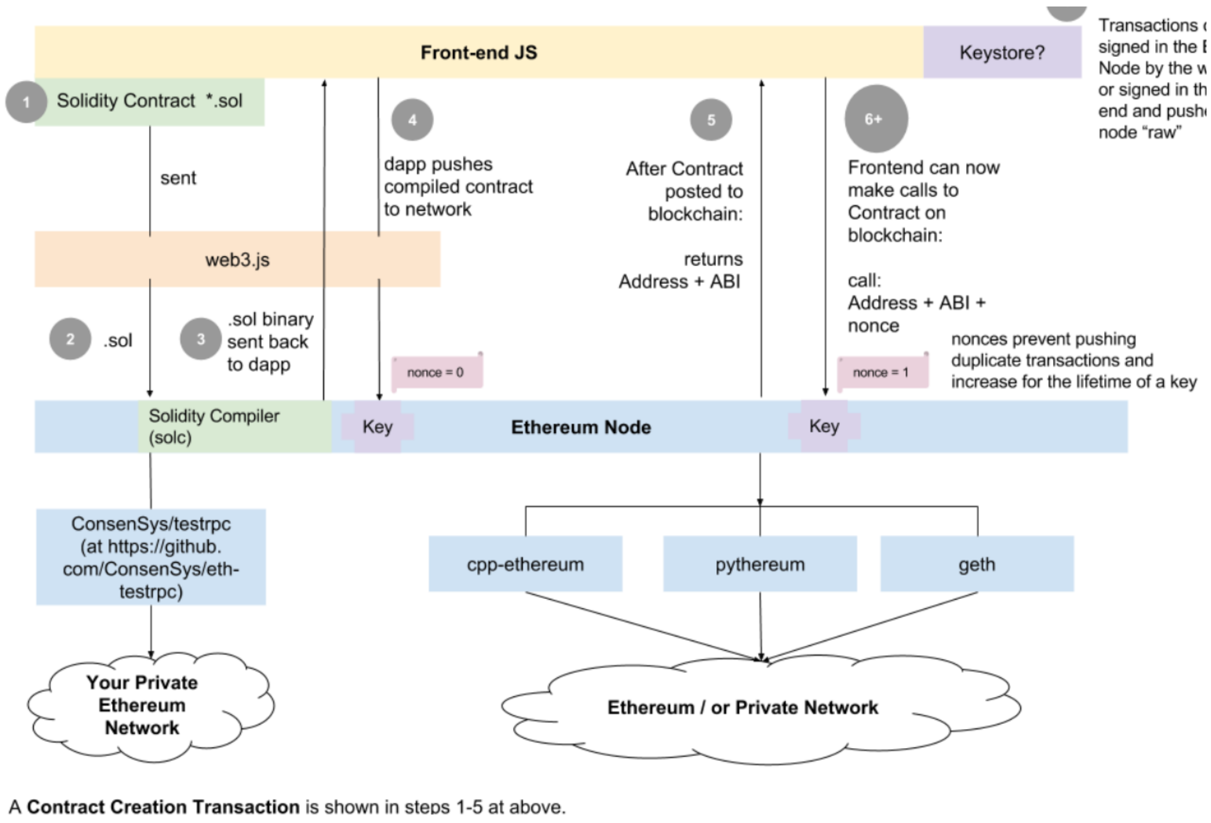

Front-end JS
Keystore?
Transactions signed in the Node by the or signed in th end and push node "raw"
1 Solidity Contract *.sol
sent
4 dapp pushes compiled contract to network
5 After Contract posted to blockchain:
returns Address + ABI
6+ Frontend can now make calls to Contract on blockchain:
call: Address + ABI + nonce
web3.js
2 .sol
3 .sol binary sent back to dapp
nonce = 0
nonce = 1
nonces prevent pushing duplicate transactions and increase for the lifetime of a key
Solidity Compiler (solc)
Key
Ethereum Node
Key
ConsenSys/testrpc (at https://github.com/ConsenSys/eth-testrpc)
cpp-ethereum
pythereum
geth
Your Private Ethereum Network
Ethereum / or Private Network
A Contract Creation Transaction is shown in steps 1-5 at above.

Figure 3.3: Web 3.0 Architecture

Figure 5 shows a range of different ways a DApp can be deployed on the Ethereum network. For the purpose of this section, we can focus on steps 1 to 4:

1. Start an Ethereum node and prepare it for development. This will be explored in greater detail under the 'Implementation' section.

2. Smart Contracts written in Solidity (or any of the bespoke ethereum languages) are compiled

3. From the binaries of the compiled contracts, deploy the contracts to the Ethereum Network or a local testnet if you are developing locally.

   "This step costs ether and signs the contract using your node's default wallet address, or you can specify another address to get back the contract's blockchain address and ABI (a JSON-ified representation of your compiled contract's variables, events and methods that you can call" [3]

4. Using the supplied web3.js library, the view can call functions from the deployed smart contracts.

3.2 Smart Contract Best Practices

3.2.1 Error Handling

One of the ways to make solidity code robust against misuse is to be generous in situations that a contract is not called properly. In the example below, a throw statement is used to handle a value out of the range that is expected of the function just as in the Java programming language:

Listing 3.1: Solidity Throw Example[8]

```
contract C {
  uint[] data;
  function append(uint item) {
    if (msg.value > 0) throw;
    data.push(item);
  }
}
```

It is important to note that unlike 'throw' in other languages, one is not able to 'catch' the resulting error message because the 'throw' statement generates an invalid JUMP in the EVM causing immediate termination.

Furthermore, 'throw' causes all gas to be spent and is thus expensive and will potentially stall calls into the current function [9]. Solidity Developer Christian Reitwiessner recommends using 'throw' in only the following two circumstances:

1. To Revert Ether transfer to the current function:

   In order to be completely certain that the function is sending back ether when called improperly, one must use throw. It is impossible to know whether the caller has enough gas to reliably process a transmission of ether back and by terminating the function, the EVM will automattically refund the funds back to the caller.

2. To Revert effects of called functions:

   If you call functions on other contracts, you can never know how they are implemented. This means that the effects of these calls are also not know and thus the only way to revert these effects is to use throw. [9]

3.2.2 Race Conditions

One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting [4]. Race conditions can occur when functions that are able to be called more than once are called repeatedly before having time to process the first invocation. We can protect against the 'spamming' of functions by throwing until the function is ready to accept a new call:

Listing 3.2: Solidity Race Condition Example[8]

```
contract C {
  bool bought;
  ...
  function buy(string name) {
    if (bought) throw;
    bought = true;
    ...
  }
}
```

3.2.3 Stopping Runaway Contracts

When deploying a contract to the Testnet or the Mainnet, it is impossible to stop it executing (as is the benefit of immutability of the EVM). However, the drawback to an immutable contract is the fact that the code cannot be edited once it has been deployed. This means that additional features have to be built into contracts to automatically mitigate the damage caused by runaway errors in production.

Consensys, the an Ethereum development agency recommend the following two solutions:

1. Circuit Breakers:

   Circuit breakers are built in methods to stop the execution of a contract.You can either give certain trusted parties the ability to trigger the circuit breaker, or else have programmatic rules that automatically trigger the certain breaker when certain conditions are met [4].

   One could set up an administrator that toggles when the contract is active:

   Listing 3.3: Solidity Circuit Breaker Example[4]

   ```
   function toggleContractActive() isAdmin public
   {
       // You can add an additional modifier
       // that restricts stopping a contract
   ```

```
        // to be based on another action,
        // such as a vote of users
        stopped = !stopped;
    }

    modifier stopInEmergency { if (!stopped) _ }

    function deposit() stopInEmergency public
    {
        // some code that cannot be run in an emergency
    }
```

2. Deliberate Delaying Mechanisms

   Even if functions are executed without tripping any circuit breakers, it is still advisable to build in some delay if possible. This allows the developers of a contract some slack to recover if malicious actions occur. A canonical use of this feature was the 27 day waiting period before funds could be withdrawn from the DAO. This ensured the funds were kept within the contract, increasing the likelihood of recovery [4]. Although it was determined that the funds were unrecoverable, it is still considered a powerful tool when combined with the techniques described above.

3.3 Summary

The three parties involved in crowdfund are the owner of the open source repository, the public who wish to see a bug fixed and finally, the developer that fixes the bug. The architectural goal of the system is to design a smart contract to facilitate the holding of a sum of Ether until the repository owner releases it to the developer or back to the funders.

The key 'best practices' for developing Ethereum smart contracts revolve around adequate error handling and the prevention of runaway contracts that have the potential to execute indefinitely.

Chapter 4

Prototype Implementation

4.1 Development Process

4.1.1 Ethereum Clients

Geth, one of the most popular client implementations of Ethereum, is written in the Go programming language. Geth was used extensively in the early phase of the project to test the mechanics of Ethereum. The Geth client is installed and controlled exclusively through the terminal.

In order to set up basic accounts and test transactions, it is not necessary to download the Ethereum blockchain nor to connect to externals as all the logic can be run locally, much like a prototyping sandbox. Within a pared down environment it was possible to rapidly test and learn the syntactic and idiomatic features of Solidity.

A benefit of using Geth to learn the basics of Ethereum is that all the transactions have to be manually mined. It forces one to learn to program in an asynchronous way in the exact manner the blockchain would be computed. An example of a transfer of a single ether between two test accounts in the geth console is shown below.

Listing 4.1: Geth Prototype Example Example[4]

```
personal.newAccount();
eth.getBalance(eth.accounts[0]);
// returns a balance of 0

miner.start(); admin.sleepBlocks(1); miner.stop();
// returns a balance greater than 0

primary = accounts[0];
secondary = accounts[1];
eth.sendTransaction({from: primary, to: secondary,
                    value: web3.toWei(1, "ether")});

// convert from wei to ether
web3.fromWei(eth.getBalance(secondary), "ether");
```

After the initial experimentation with Solidity in Geth, the project switched to a python implementation of the Ethereum client called TestRPC. It is optimized for automated testing (i.e. mining) as opposed to stepping through transactions manually with Geth. TestRPC comes with 10 pre-funded accounts without passwords and a host of functions that allow for further automation.

4.1.2 Frameworks and Tools

Truffle is a "development environment, testing framework and asset pipeline for Ethereum" [5]. With truffle, the creation, testing and deployment of a project is made significantly easier. Every new project created with truffle comes with a skeleton directory and pre-configured files that help with compilation of the contracts. The project maintained the basic structure of a truffle project file and leveraged the built in connections the Solidity contracts had with the front-end. Furthermore it works extremely well with TestRPC allowing for rapid deployment of the contracts to the local blockchain and iteration of the contracts.

After truffle compiles a solidity contract it is deployed to the network (either the local TestRPC or an external network). Once a contract is on the network, web applications can use the web3.js API interact with them. web3.js is simply a javascript library that allows higher level functions calls to be made to deployed smart contracts. Examples of these functions can be found at the official documentation (https://github.com/ethereum/wiki/wiki/J
API).

The project was coded using the Atom text editor due to the fact that Solidity linting packages were available as well as its extensive native git integration.

4.1.3 Workflow

The workflow for prototyping the DAPP consisted of the following four steps:

1. Starting the Ethereum node (in our case it was testRPC)
2. Compiling the smart contracts. This is done by calling the command 'truffle compile'.
3. Deploying the DAPP. This is done by calling the command 'truffle deploy'.
4. Interacting with the smart contracts on the network. Each truffle project has an app.js file encapsulates the logic necessary for the web app calling contracts using web3.js.

4.2 Solidity Contracts

4.2.1 Realized Architecture

The implementation of the smart contracts and the corresponding distributed application mirrored the design closely. The Factory.sol contract is responsible for both the generation of Crowdfund.sol contracts as well as to act as an interface with the DAPP (see Figure 7. on the following page). The DAPP refers to the project as a whole i.e. the deployed smart contracts and the front-end website that the users interact with. The front-end comprises of html, css and javascript files designed to be as light and simple as possible. The javascript files call the web3.js library to call the deployed smart contracts.

Figure 6. explains the relationship between the Factory.sol contract and the Crowdfund.sol contracts in greater detail. As proposed in the Design section, the Factory.sol

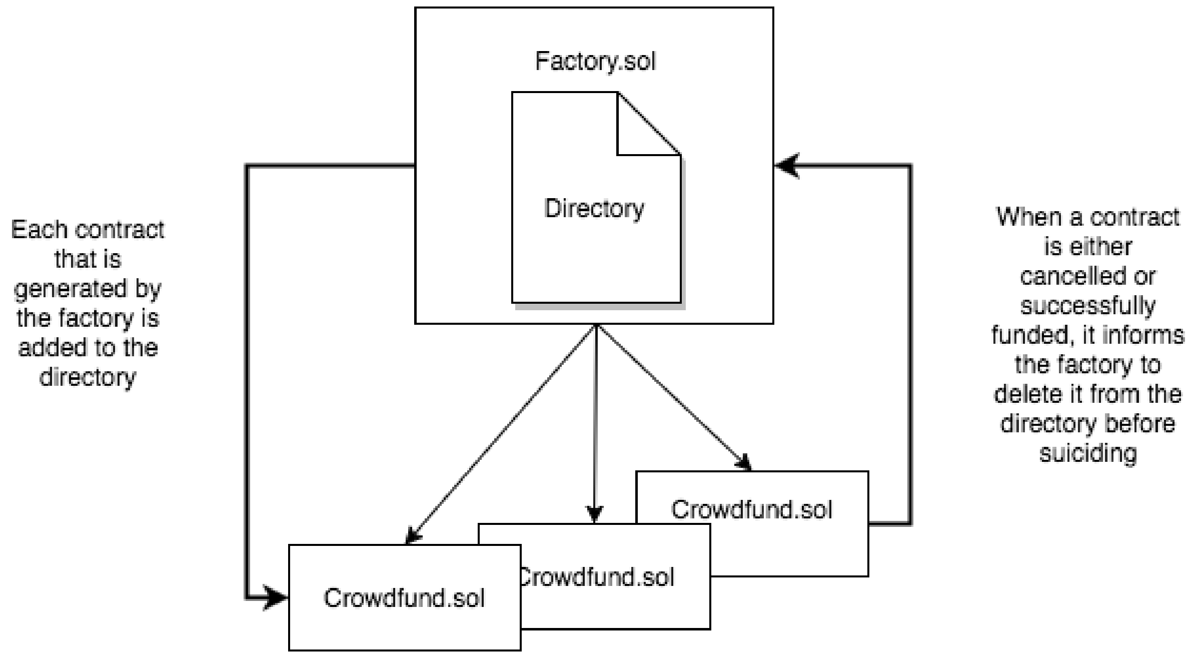

Factory.sol
Directory
Each contract that is generated by the factory is added to the directory
When a contract is either cancelled or successfully funded, it informs the factory to delete it from the directory before suiciding
Crowdfund.sol
Crowdfund.sol
Crowdfund.sol

Figure 4.1: Relationship Between Factory.sol and Crowdfund.sol

contract contains a directory of all Crowdfunds its created. This reduces complexity and thereby encourages decentralization on the front end as any front-end (not necessarily the one supplied by the project) need only know about the Factory to access all active Crowdfunds. Furthermore, having a centralized directory prevents fraud by allowing the public to cross-check Crowdfunds to see if they were indeed created by the project and not by another system.

For a Crowdfund to be removed from the directory, the Crowdfund would have to do it itself since the point of the project was to automate the disbursement of funds without the funders or the project owner needing to do more than either fund or signal the end of a contract. This means that each Crowdfund.sol would need to know about the Factory.sol.

In addition, for a Crowdfund to call a contract, a Factory object would need to be created (this will be explicated in a later subsection) and therefore would need to import the class. However, this throws a cyclic inheritance error during truffle compilation. This was resolved by employing an abstract class of Factory.sol and importing that into each Crowdfund.sol as shown in Figure 8.

4.2.2 Factory.sol

Factory.sol was also implemented very closely to how it was designed. The functions in Factory.sol fall into two categories: manipulation of Crowdfund.sol contracts in its directory and 'getters' for data on Crowfund.sol contracts in said directory.

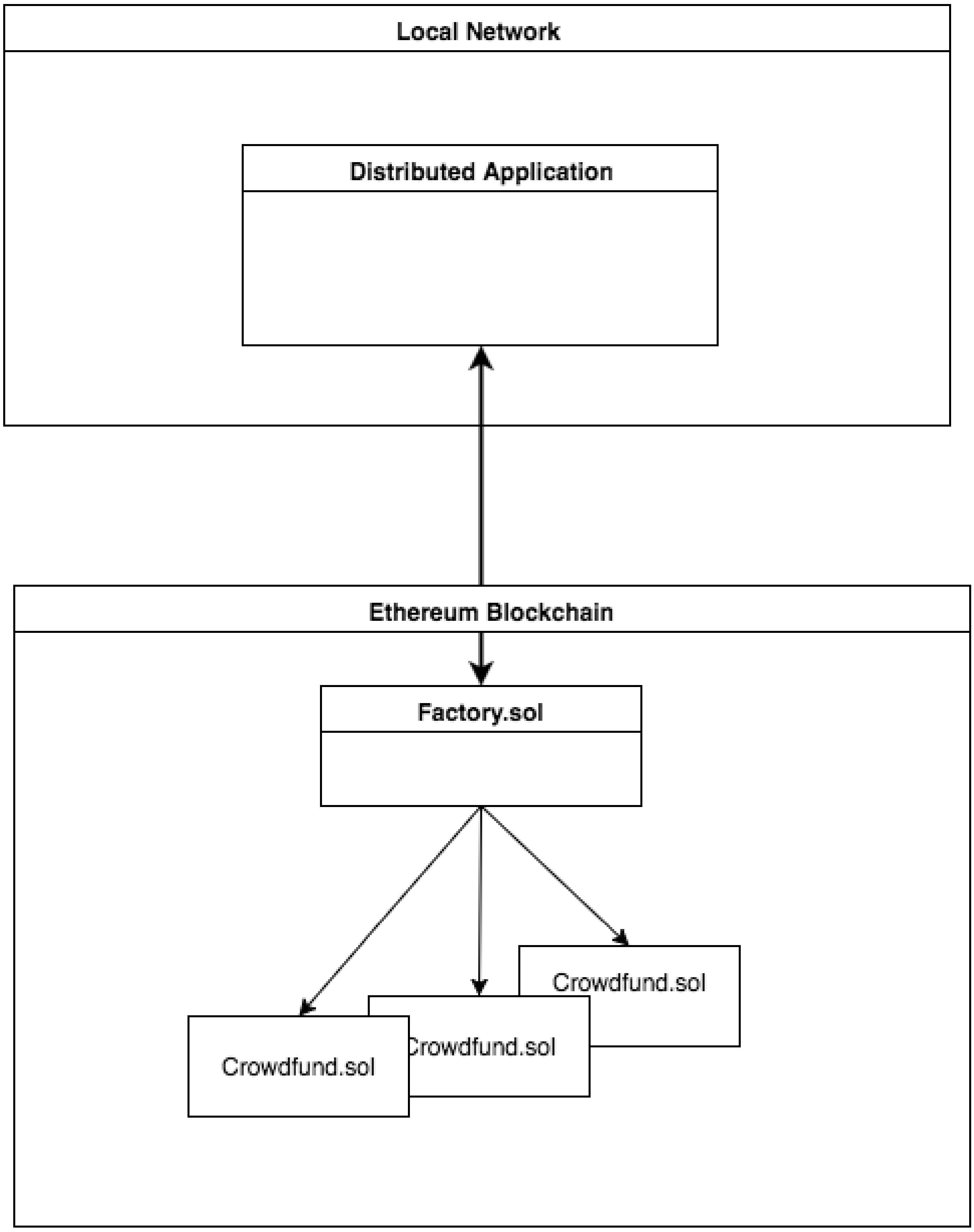


Figure 4.2: High-level Deployed Architecture

Since truffle does not allow parameters for object contstructors, the attributes for each Crowdfund.sol had to be set after each Crowdfund generation:

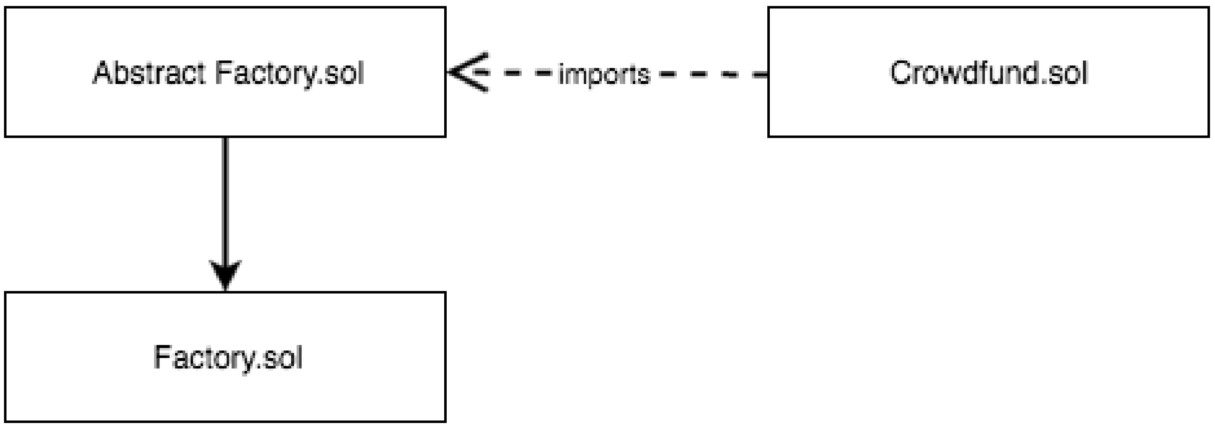


Figure 4.3: Class Diagram

Listing 4.2: geth Prototype Example Example[**?**]

```
function createContract (address _repo_owner,
    string _repo_web_address) returns (address){

    address crowdfund_address = new Crowdfund();

    // Set initial conditions
    Crowdfund crowdfund = Crowdfund(crowdfund_address);
    crowdfund.setRepoOwner(_repo_owner, _repo_web_address);
    crowdfund.setIndex(_index);
    crowdfund.setOrigin();
}
```

Another optimization that was implemented was a custom (albeit, very simple) method to delete an entry from the directory. In most high-level languages it is trivial to remove a an entry from an array however when running methods on the Ethereum network it is imperative that as few computing resources are used to minimize the gas cost of calling the function. Hence, when a contract is removed from the directory, it is simply swapped with the last element and the length of the contract is decremented. If the element is the last in the array, the contract is decremented without any swaps:

Listing 4.3: geth Prototype Example Example[**?**]

```
  function deleteContract(uint index) returns (bool){

    if (index != crowdfunds.length -1){
      CrowdfundStructure temp = crowdfunds[crowdfunds.length -1];
      crowdfunds[crowdfunds.length -1] = crowdfunds[index];
      crowdfunds[index] = temp;
    }
    delete crowdfunds[crowdfunds.length-1];
    crowdfunds.length--;
    return true;
  }
}
```

4.2.3 Crowdfund.sol

Similar to Factory.sol, Crowdfund.sol's implementation matched its design extremely closely. The two core functions of sending the bounty of a crowdfund to a specified address and the cancelling a crowdfund (thereby releasing funds back to the founders proportionate the the amount they contributed) are presented below:

Listing 4.4: geth Prototype Example Example[**?**]

```
/* If the agent decides the crowdfund is successful it is closed */
function sendTo(address recepient){
if (msg.sender == repo_owner){
  destroySelf();
  suicide(recepient); // Send all funds to seller
}
else
  throw;
}

/* Cancel the crowdfund and return money to funders*/
function cancel() {
if (msg.sender == repo_owner){
  for (uint i = 0; i < funders.length; ++i) {
    funders[i].addr.send(funders[i].amount);
  }
  destroySelf();
  suicide(repo_owner); //delete the contract and
                       //give whatever money is leftover
                       //to the owner of the repository
}
else
  throw;
}
```

Note that in line with Smart Contract best practices, errors are thrown to cease the function call without any side effects.

4.2.4 Testing

One of the main advantages of truffle is the automated testing suite that is bundled with the framework. During development, draft implementations of the contracts were periodically tested via the automated testing suite which handled the compilation, deployment and interaction of the contracts. The truffle test suite is modelled on some of the most popular modern javascript test suites such as jasmine and is alike in both appearance and functionality except for that it is optimized to call the web3.js library when executing the tests. Each test is prefaced by an 'it' function that describes what the test does. The functions are then chained together in the form of javascript promises. This ensures that the previous function has been completed (as the blockchain operates asynchronously) before calling the next ones. Assert functions are used to check an expected value matches the one outputted by a test. A simple test used in production to check if deleting a contract from a directory is shown below:

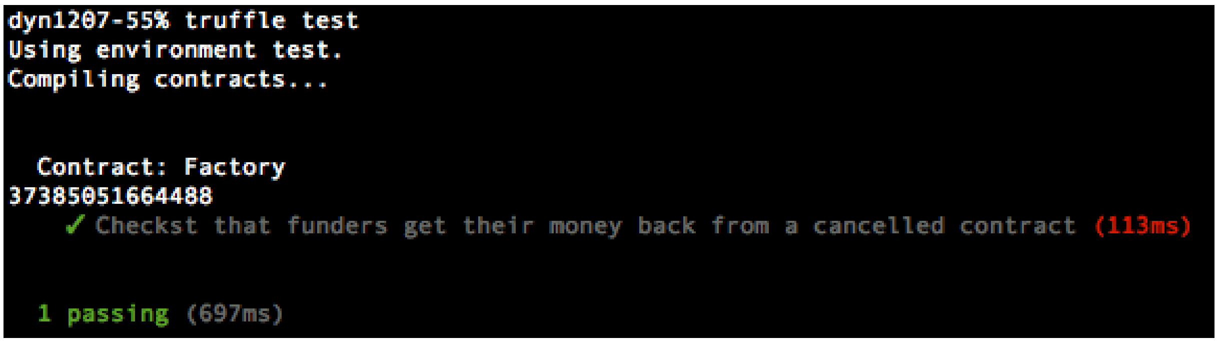
Listing 4.5: geth Prototype Example Example[**?**]

```
it("Creates a contract then deletes it", function(){

var fact;
var cf;

return Factory.new({from: accounts[0]})
  .then(function(factory){
    fact = factory;
    return factory.getNumberOfContracts.call();
  }).then(function(len){
      assert.equal(len.toNumber(), 0, "Array should be empty")
      return fact;
  }).then(function(factory){
    factory.createContract(accounts[0], "moscow-road",
    { from: accounts[0] })
    return factory;
  }).then(function(){
    return fact.deleteContract(0, {from: accounts[0]});
  }).then(function(){
    return fact.getNumberOfContracts.call();
  }).then(function(len){
    assert.equal(len.toNumber(), 0, "Array should be empty")
  });
});
```

Figure 4.4 shows the output produced by the console.



```
dyn1207-55% truffle test
Using environment test.
Compiling contracts...


  Contract: Factory
37385051664488
    ✓ Checkst that funders get their money back from a cancelled contract (113ms)


  1 passing (697ms)
```

Figure 4.4: Output from truffle test suite

4.3 Summary

The smart contract design that was decided on involves two types contracts: a factory contract and the crowdfund contracts it generates. Only one copy of the factory will be created and it will also serve as a directory for a user interface to query for all the crowdfunds in existence on the network. Each crowdfund contract will contain all the necessary logic to initiate and close a crowdfund, thereby limiting errors and risk to only that particular contract.

Chapter 5

Deployment To The Ethereum Network

5.1 Ethereum Networks

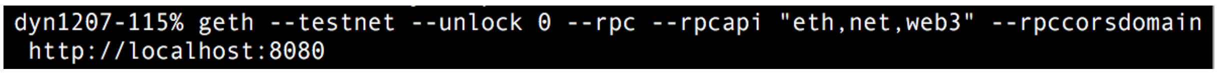
The Ethereum Foundation has developed two versions of Ethereum: the Mainnet and the ‘Morden’ Testnet. The Mainnet is the ‘main network’ and is often simply called ’The Ethereum Network’. In terms of code, there is no difference between the Mainnet and the Testnet. Simply because one of the implementations has been designated as the primary network, the vast majority of miners have decided to mine on the Mainnet thus increasing the price of ether and the difficulty of mining additional blocks.

The Testnet has been designed to run in parallel to the Mainnet and be upgraded simultaneously. Therefore, any applications deployed on the Testnet can be immediately transferred to the Mainnet without any difference in functionality except for a higher gas price cost.

Similar to the description given under the ’Implementation Section’, the creation of a distributed application requires two phases:

1. the deployment of ethereum contracts to a blockchian (either a test blockchain in memory i.e. testrpc) or a blockchain on a network being maintained by external nodes (Either the Mainnet or the Testnet).
2. the ’serving’ of the application that utilizes the web3.js javascript library to interface with the blockchain the contracts are deployed on and the user who interacts with front end logic (html, css).

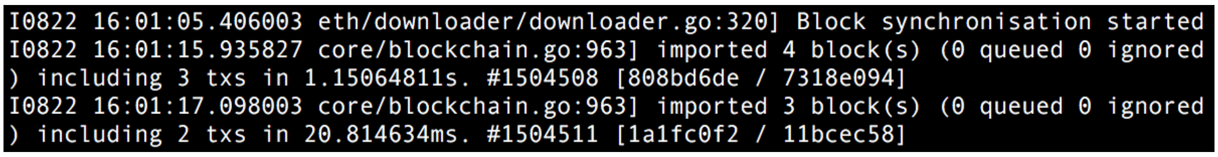
Using the Geth client, a local account is unlocked and the local machine is used as a node in the Ethereum Testnet using the follwoing command:



```
dyn1207-115% geth --testnet --unlock 0 --rpc --rpcapi "eth,net,web3" --rpccorsdomain
 http://localhost:8080
```

Figure 5.1: Starting a node on the Ethereum Testnet

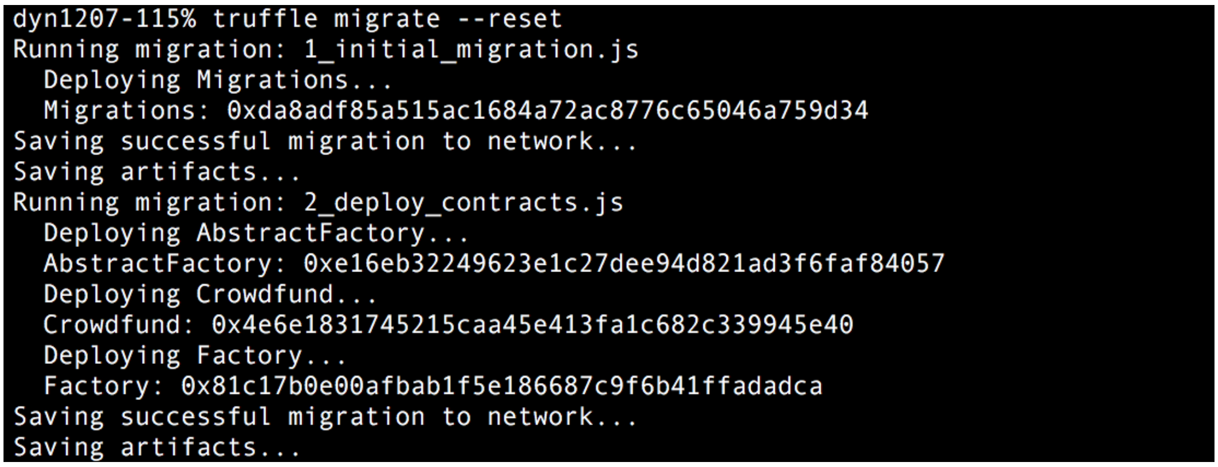
Once a node is running it downloads the entire blockchain from the network. Geth then updates the local version of the blockchain with new blocks as they are propagated throughout the network. This is called ’block synchronisation’:



```
I0822 16:01:05.406003 eth/downloader/downloader.go:320] Block synchronisation started
I0822 16:01:15.935827 core/blockchain.go:963] imported 4 block(s) (0 queued 0 ignored
) including 3 txs in 1.15064811s. #1504508 [808bd6de / 7318e094]
I0822 16:01:17.098003 core/blockchain.go:963] imported 3 block(s) (0 queued 0 ignored
) including 2 txs in 20.814634ms. #1504511 [1a1fc0f2 / 11bcec58]
```

Figure 5.2: Block Synchronisation

The truffle framework automates the majority of the work needed to deploy a contract to the network. All that is necessary is a command to point the migration function to the Testnet as opposed to the testrpc. Note that a Testnet node has to be active on the machine one is deploying from. The strings after each of the smart contracts shown below refer to the memory address on the Ethereum Testnet blockchain to where the contracts are compiled and saved to.



```
dyn1207-115% truffle migrate --reset
Running migration: 1_initial_migration.js
  Deploying Migrations...
  Migrations: 0xda8adf85a515ac1684a72ac8776c65046a759d34
Saving successful migration to network...
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying AbstractFactory...
  AbstractFactory: 0xe16eb32249623e1c27dee94d821ad3f6faf84057
  Deploying Crowdfund...
  Crowdfund: 0x4e6e1831745215caa45e413fa1c682c339945e40
  Deploying Factory...
  Factory: 0x81c17b0e00afbab1f5e186687c9f6b41ffadadca
Saving successful migration to network...
Saving artifacts...
```

Figure 5.3: Deploying contracts to the Testnet

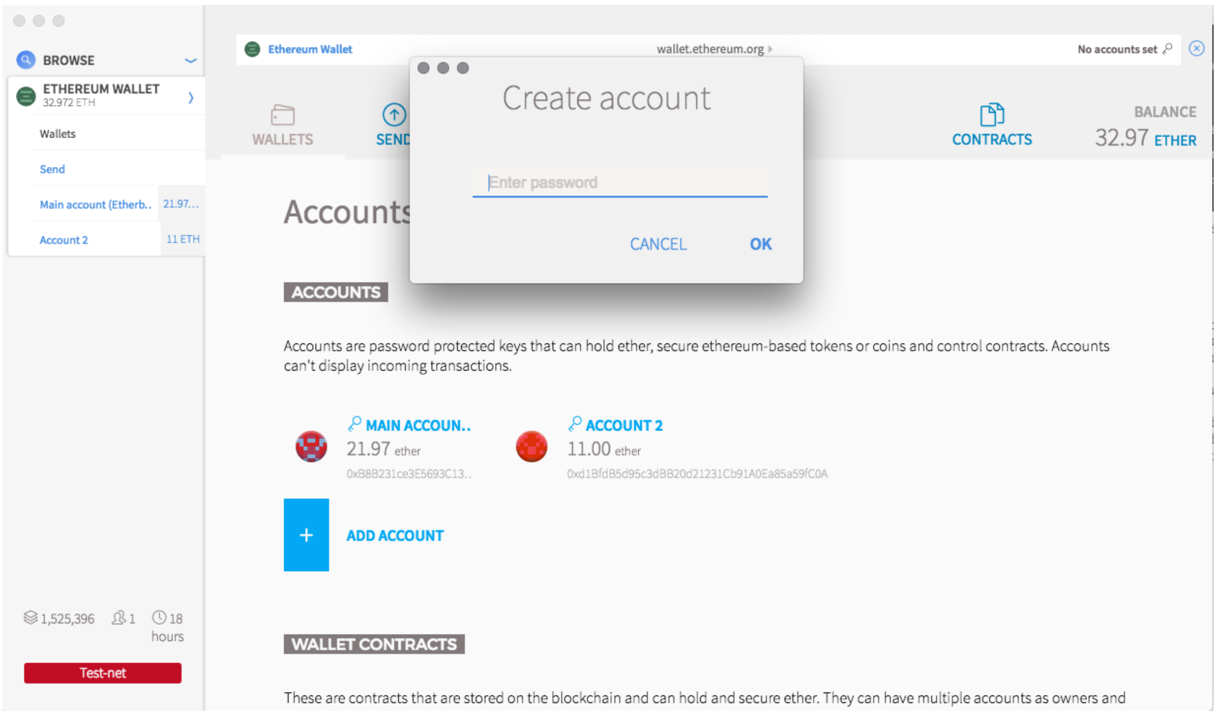
5.2 The Mist Browser

Released and supported by the Ethereum Foundation, the Mist Browser is technically the first Web 3.0 as it contains the functionality of a standard web browser with features geared to interactions with the Ethereum Networks.

The Mist browser can be thought simply as a graphical user interface for the ethereum blockchain. The Mist browser runs its own Geth client (so it is important to stop the any other nodes running at the same time) and the interface allows users to execute all the the command line functionality but with the ease of having buttons and images, thereby significantly lowering the barrier of entry for users to interact with distributed applications.

For example, the creation of an Ethereum wallet is done through a wizard and standard functions are shown immediately to the user such as depositing ether. Additional functionality not avaliable on the command line is also shown such as generating a QR code for the wallet address (Figure 5.13).

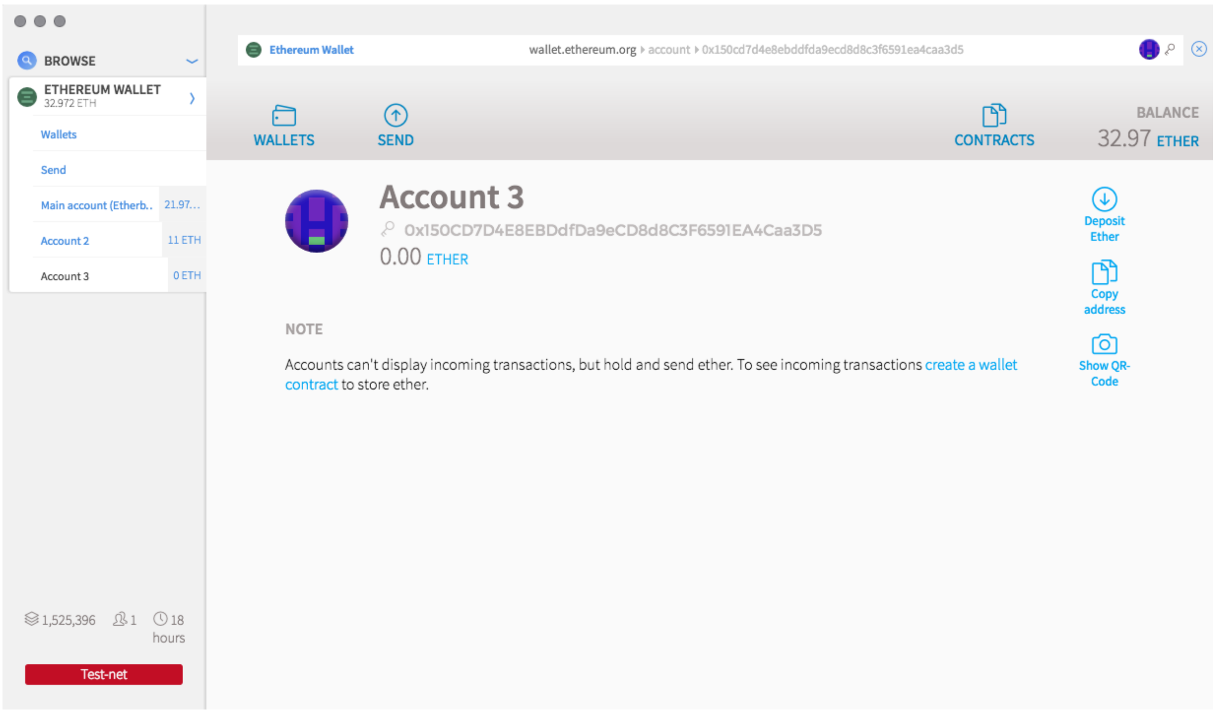
Mist also functions as a working Web 2.0 browser that allows the user to accesss standard web pages such as google.com. The benefit is that any front-end interfaces blockchain based-applications that hosted on a typical Web 2.0 servers can be interacted with through Mist but additional actions such as the transfer of ether can be done with Mist.



Accounts are password protected keys that can hold ether, secure ethereum-based tokens or coins and control contracts. Accounts can't display incoming transactions.

These are contracts that are stored on the blockchain and can hold and secure ether. They can have multiple accounts as owners and

Figure 5.4: Mist Wallet Creation part 1



Accounts can't display incoming transactions, but hold and send ether. To see incoming transactions create a wallet contract to store ether.

Figure 5.5: Mist Wallet Creation part 2

5.3 etherscan.io

Etherscan (www.etherscan.io) is a web application that provides information on the Ethereum Mainnet and Testnet. Etherscan describes itself as a "Block Explorer, Search, API and Analytics Platform for Ethereum, a decentralized smart contracts platform." [**?**].

Etherscan was a crucial part of the deployment process because it was used to examine the Testnet's blockchain and see that the contracts were deployed correctly. Each contract is deployed onto a particular address on the blockchain and it is possible to retrieve and inspect the each latest block for new contracts.

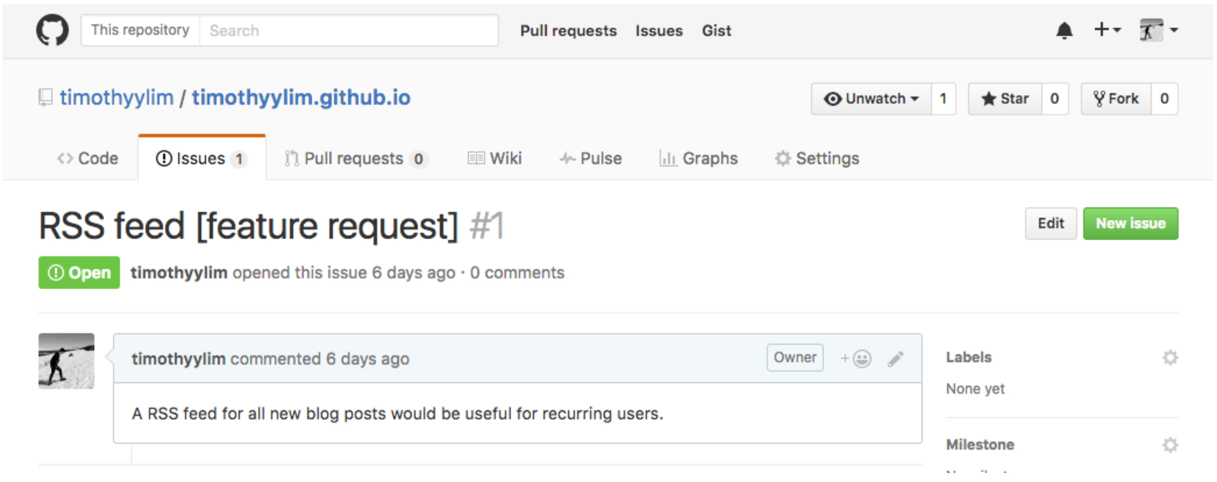
5.4 User Walkthrough

In order to demonstrate the advantages of the Mist browser, it is useful to walkthrough the first user story defined in the 'Design' chapter.

1. A proposal is successfully funded:

   The owner of the repository merges the pull request into the main branch, or any other branch in production and the developer is paid the bounty. The code is crowdfunded by the public and by being merged to repository, is available to be used by the public.

The following images represent the process through which the project satisfied the first user story. An explanation of each step is detailed in the captions with the corresponding images.



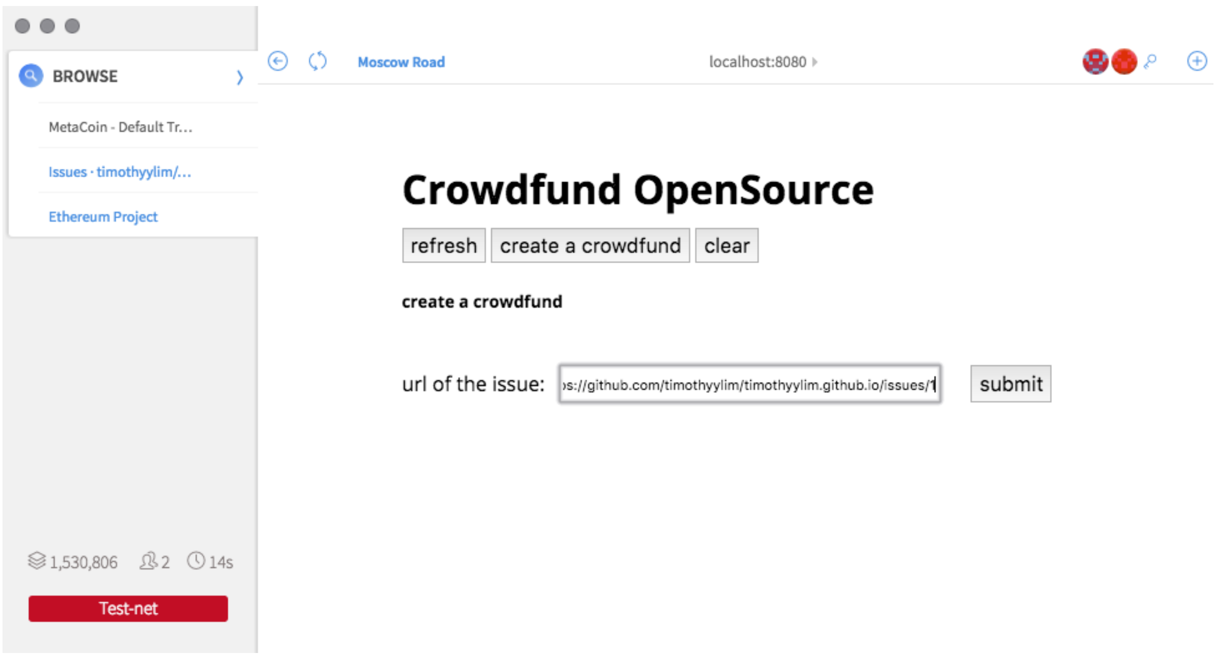
RSS feed [feature request] #1

timothyylim opened this issue 6 days ago · 0 comments

timothyylim commented 6 days ago

A RSS feed for all new blog posts would be useful for recurring users.

Figure 5.6: A feature request is created in a public repository on github



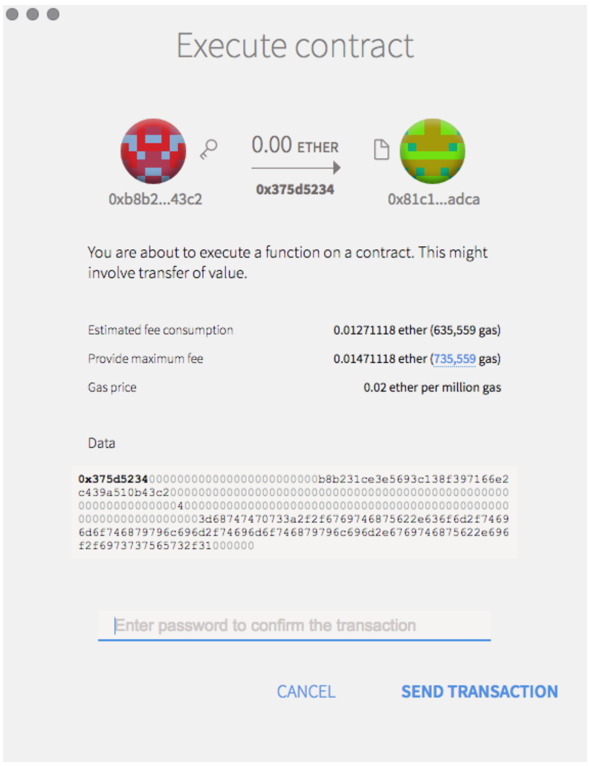
Crowdfund OpenSource

refresh | create a crowdfund | clear

create a crowdfund

url of the issue: s://github.com/timothyylim/timothyylim.github.io/issues/1 submit

Figure 5.7: The user then navigates to the deployed distributed application with the Mist browser. For now it is being hosted temporarily from a local host but in the future it is hoped that the front-end can exist on a Web 2.0 server. The user inputs the url from the github issue and clicks submit.



You are about to execute a function on a contract. This might involve transfer of value.

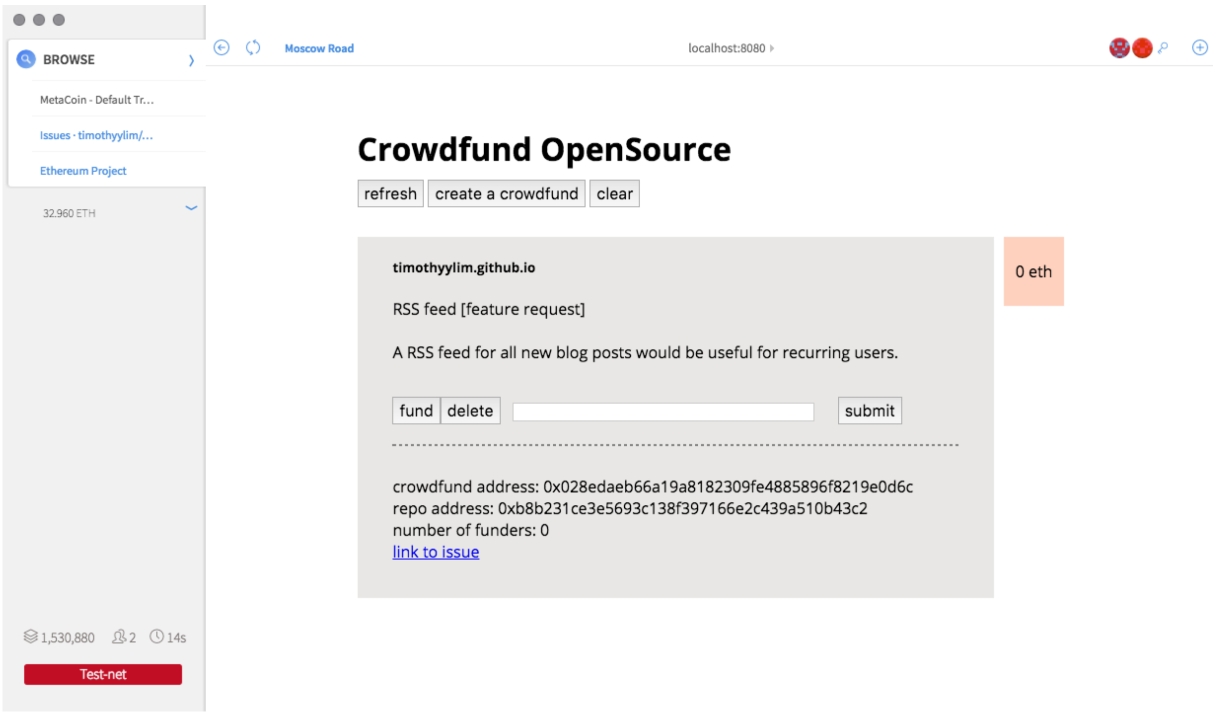
Estimated fee consumption	0.01271118 ether (635,559 gas)
Provide maximum fee	0.01471118 ether (735,559 gas)
Gas price	0.02 ether per million gas

Data

```
0x375d5234000000000000000000000000b8b231ce3e5693c138f397166e2
c439a510b43c2000000000000000000000000000000000000000000000000
000000000000004000000000000000000000000000000000000000000000
000000000000000003d68747470733a2f2f6769746875622e636f6d2f7469
6d6f746879796c696d2f74696d6f746879796c696d2e6769746875622e696
f2f6973737565732f31000000
```

Enter password to confirm the transaction

Figure 5.8: To create a contract, it requires some ether to write the new information to the blockchain. As shown by the dialog pop-up generated by Mist, it costs a negligible amount of Ether.



Crowdfund OpenSource

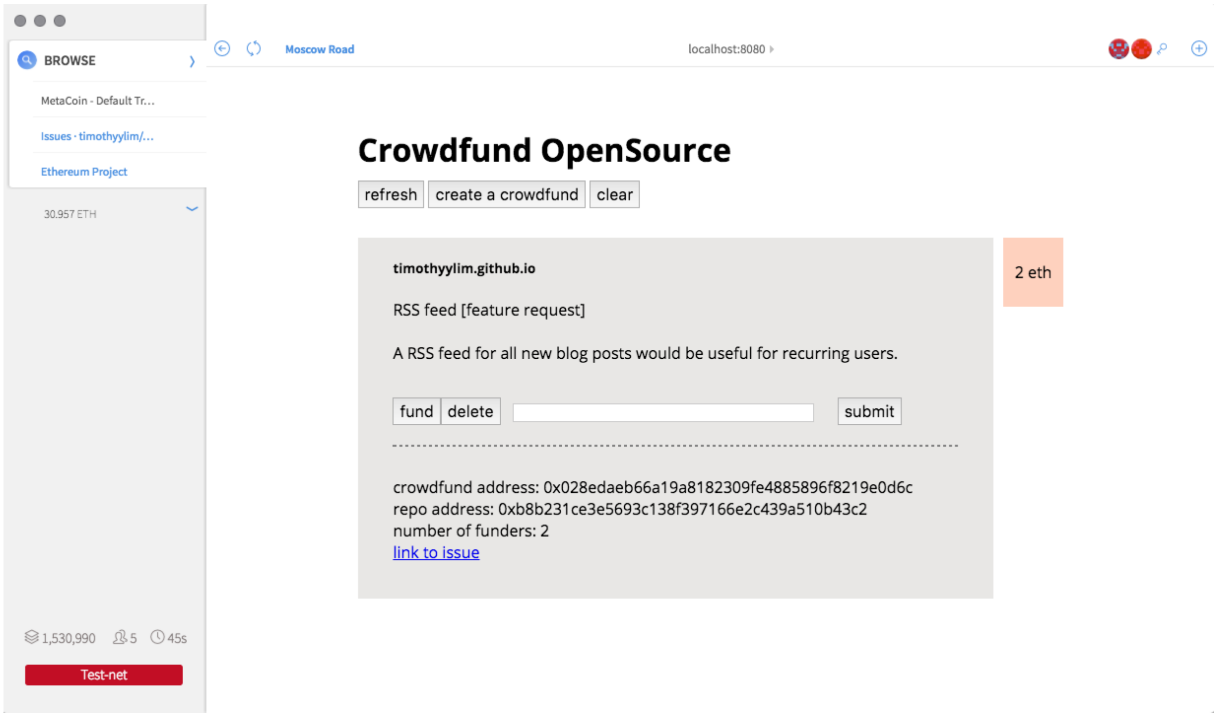
timothyylim.github.io

RSS feed [feature request]

A RSS feed for all new blog posts would be useful for recurring users.

crowdfund address: 0x028edaeb66a19a8182309fe4885896f8219e0d6c
repo address: 0xb8b231ce3e5693c138f397166e2c439a510b43c2
number of funders: 0
link to issue

Figure 5.9: An crowdfund for the contract is generated. The text from the github issue is displayed as well as the url that it is derived from. Additional meta-data is shown below the dotted line such as the address for the crowdfund on the Ethereum blockchain.



Crowdfund OpenSource

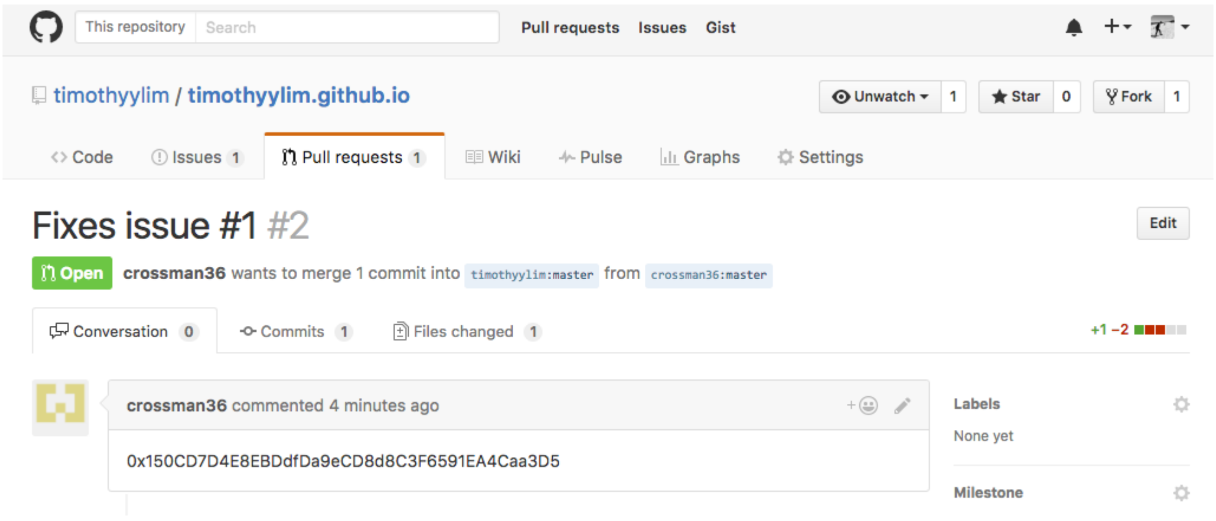
timothyylim.github.io

RSS feed [feature request]

A RSS feed for all new blog posts would be useful for recurring users.

crowdfund address: 0x028edaeb66a19a8182309fe4885896f8219e0d6c
repo address: 0xb8b231ce3e5693c138f397166e2c439a510b43c2
number of funders: 2
link to issue

Figure 5.10: When a user clicks on the 'fund' button it, by default, funds the contract to the value of a single Ether. For demonstration purposes, the button has been clicked twice and the bounty for the issue has been shown to be raised to 2 Ether.



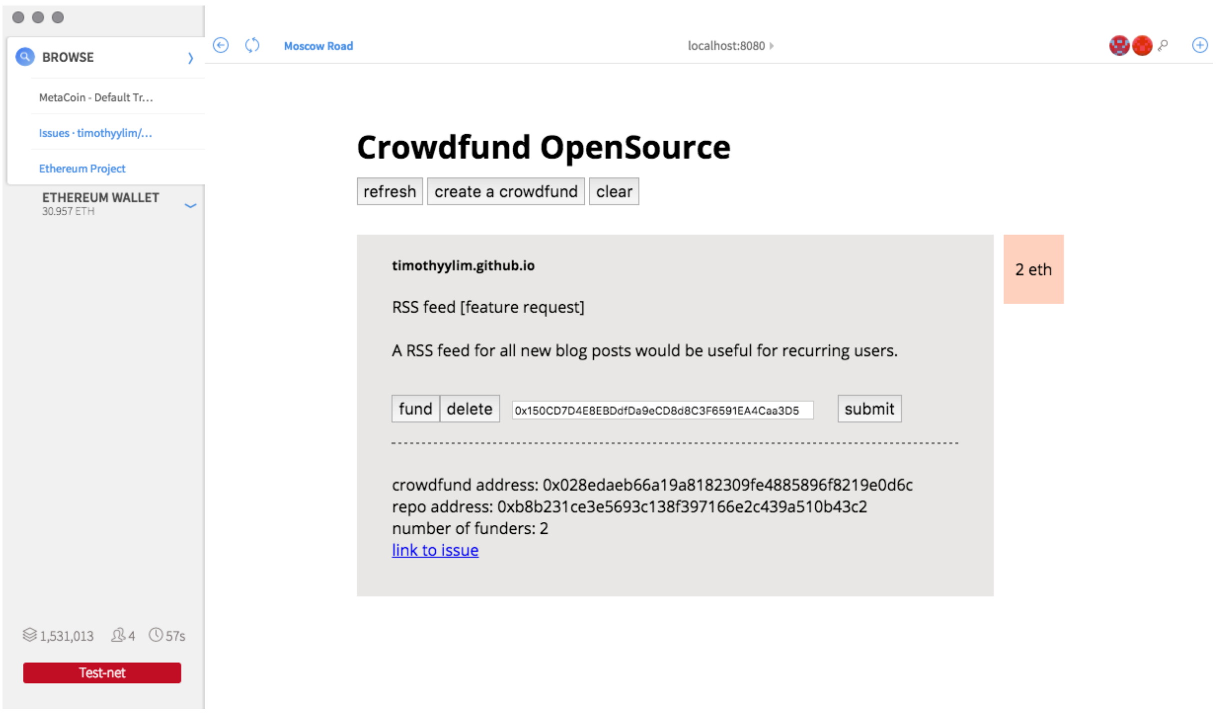
Fixes issue #1 #2

Open crossman36 wants to merge 1 commit into timothyylim:master from crossman36:master

crossman36 commented 4 minutes ago

0x150CD7D4E8EBDdfDa9eCD8d8C3F6591EA4Caa3D5

Figure 5.11: Now a developer can submit a pull request to fix the issue and append his or her ethereum wallet address to the pull request in order to claim the bounty.



Crowdfund OpenSource

timothyylim.github.io

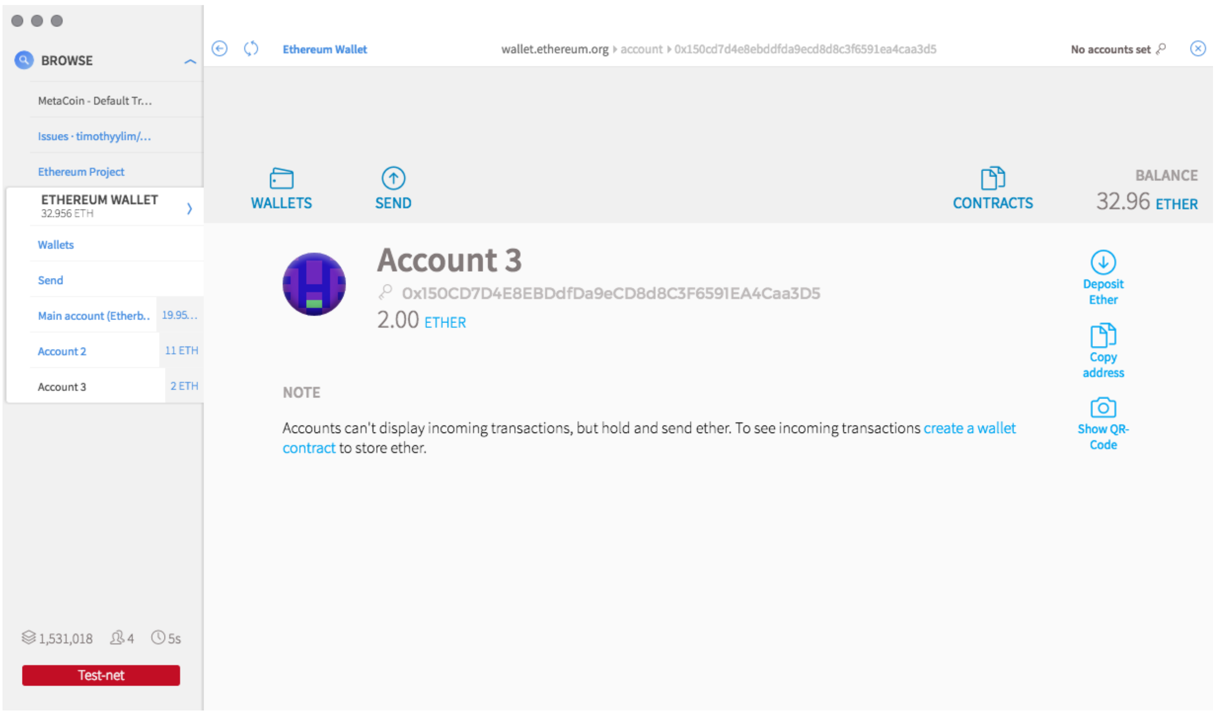
RSS feed [feature request]

A RSS feed for all new blog posts would be useful for recurring users.

0x150CD7D4E8EBDdfDa9eCD8d8C3F6591EA4Caa3D5

crowdfund address: 0x028edaeb66a19a8182309fe4885896f8219e0d6c
repo address: 0xb8b231ce3e5693c138f397166e2c439a510b43c2
number of funders: 2
link to issue

Figure 5.12: If the repository owner is satisfied with the pull request and wishes to payout the bounty to the developer, the wallet address is simply typed into the textbox and the submit button is clicked. This destroys the contract, removes it from the directory in the Factory and pays out the bounty to the address given.



Account 3

0x150CD7D4E8EBDdfDa9eCD8d8C3F6591EA4Caa3D5

2.00 ETHER

NOTE

Accounts can't display incoming transactions, but hold and send ether. To see incoming transactions create a wallet contract to store ether.

Figure 5.13: The wallet has now gained 2 Ether from claiming the bounty

5.5 Summary

Ethereum has two functioning networks running two sepearte blockchains, the Mainnet and the Testnet. Functionally, they are identical and the only difference between them is the mining difficulty of Ether. For this reason, development and deployment was focused on the Testnet, being the cheaper of the two. Any application created to the specification of the Testnet can be ported to the Mainnet given the required Ether.

A node was set up locally and the Testnet blockchain was downloaded. From there, a browser called 'Mist' was used to interact with the platform and to mine ether and manage the acocunts. A walkthrough of the basic features is presented.

Chapter 6

Evaluation

6.1 Review of Objectives

As defined in the introductory chapter, the objectives for this project were the following:

1. Introduce the concept of blockchain-based networks and a high level understanding of Ethereum
2. Analyse and capture the key user stories from existing crowdfunding platforms and rework them into the an Ethereum context
3. Develop a prototype application: Encode the logic of a crowdfunding platform into a proof of concept that can run on a local simulation of the Ethereum Network
4. Deploy the prototype to the Ethereum Network: Ensure the basic functionality can be computed and verified by external anonymous nodes and that the application is 'live'
5. Evaluate the feasibility of the prototype and examine its effectiveness compared to existing solutions

The first two objectives of introducing the notion of blockchains, their relation to the Ethereum Network and their potential for serving as a platform for a decentralized crowdfunding application are satisfied in the Background and Design chapters.

With regards to the objective to develop a prototype, out of the three user stories that were gathered in the 'Design' chapter, only the first two were successfully implemented in the smart contracts:

1. A proposal is successfully funded:

   The owner of the repository merges the pull request into the main branch, or any other branch in production and the developer is paid the bounty. The code is crowdfunded by the public and by being merged to repository, is avaliable to be used by the public.

2. A proposal fails to be funded:

   If a proposal cannot raise enough money to fund a solution it should be withdrawn and all the money should be returned to the funders. In order to not have funds sitting in a smart contract indefinitely it makes sense to have a timeout feature (either a hard deadline of 30 days or perhaps set by the funders).

3. A proposal is successfully funded but is not approved by the maintainers of the project:

   Even if the code is written, there is no guarantee that the maintainers of a project are going to accept it. This could be due to security reasons or to prevent bloating the codebase. There are two ways to resolve this problem, either each proposal has to be vetted by the repository owners before funding begins or the platform can give the repository owners the ability to cancel any crowdfund at anytime. The latter option seems more suitable for a basic prototype.

Due to time limitations, it was difficult to create the functionality for contracts to suicide after a certain amount of time which was the final user story.

Although it was possible to 'ask' contracts for the time that has elapsed since its creation no solution was found to automate this query in the blockchain. A standard Web 2.0 solution would be to run a cron job using UNIX time-stamping however, this defeats the main purpose of building a distributed application atop Ethereum which will have guaranteed success of execution.

The final objective of deploying the prototype was fulfilled. The combination of the truffle framework and the testrpc library was crucial in quickly iterating versions of the contracts and the application as a whole. There were some issues with porting the project from a local testrpc environment and deploying the contracts to the testnet but it was made much easier given the avaliability of the Mist browser.

The two advantages of an Ethereum-based crowdfunding platform over existing Web 2.0 platforms such as Bounty Source was that it was a) 'Unhackable' and b) Allowed funds to be transferred for negligible costs. While proving the immutability of the proof-of-work algorithm powering the Ethereum network is beyond the scope of this paper, the deployment of the application is a tangible (if somewhat simplistic) innovation in terms of developing the technology of crowdfunding platforms.

The costs of the main functionality on the platform is summarized below

User Action	Price in Ether	Price in U.S. Dollars (28th August 2016)
Creation of A Crowdfund	0.012	$0.13
Funding of 1 Ether	0.00167	$0.0182
Paying out A Crowdfund	0.001	$0.01094
Cancelling A Crowdfund	0.001	$0.01094

6.2 Future Work

6.2.1 Major Bug Fixes

There is a currently a major bug in the logic behind the updating of the directory. When a crowdfund is removed (either because it has been payed out or cancelled), the directory does not update correctly and does not decrement its length. This bug was not caught until quite late the development process because it does not occur when the last index is removed.

The second major bug that has yet to be fixed is the loss of pointers to accounts in the Mist browser. When the application is first loaded, the javascript on the front end interacts with the Mist browser's api and collects information on the unlocked accounts.

However, when the page is refreshed the pointers to the accounts are lost and a reference error occurs. This can be solved by calling the function to access the accounts every time the refresh button is clicked but more efficient solutions should be explored.

6.2.2 Implementation of Contract Timeouts

The third user story was to implement a function for a contract to suicide after a period of thirty days. The purpose of a timeout was to give assurance to funders that if an issue was not resolved that they would eventually be given their money back.

The problem, as illustrated above, was that Ethereum contracts can never 'call' on information that is not contained in the blockchain. This is because a history of any blockchain has to be mathematically reproducible. A potential solution is to use a smart contract called Ethereum Alarm clock (www.ethereum-alarm-clock.com) that incentivises agents in the network to send signals at a given time to addresses on the blockchain. The features of Ethereum Alarm Clock are the following:

1. Function calls can be scheduled to be executed against any contract
2. Scheduling can be done by contracts or ethereum account holders.

The Ethereum Alarm Clock can be loaded with Ethereum ahead of time and the address of the Factory can be specified as the address to which the Alarm Clock will send a transaction every thirty days. The Factory can set a default function to receive Ether from the Ethereum Alarm Clock and suicide itself after checking that it is from the Ethereum Alarm Clock smart contract address.

6.2.3 History of crowdfunds/logging facility

A key feature that is missing from the prototype crowdfunding platform is a public history of crowfunds adn their associated donors and repository owners. In order for funders and developers to gauge the responsiveness of open source maintainers it is crucial that data of previous campaigns is made available.

In addition, a reputation system can be built up amongst all three parties of the ecosystem. Funders can be given a score on how much and how often they fund open source projects. This could be a badge of credibility to companies who wish to be seen as supporting open source products that they use in their for-profit systems. Developers who have a good track record with pull requests can be identified by repository owners and funders and potentially good workers for a project. Finally, maintainers of open source projects can be tracked for their reponsiveness to developer queries and their propensity to pay out rewards without conflict.

The current implementation of the Crowdfund.sol smart contract specifies that a contract is 'suicided' and removed from the blockchain at the end of its lifetime. It is possible to add functionality for the contract to add itself to an array in the Factory.sol contract that would act as a historical reference for all contracts that the Factory had created. In addition, meta-data about the lifetime of the contract can be encoded into the data to provide a richer database to draw from.

6.2.4 A/B Testing

Alpha beta tests are randomized tests to gauge whether a new feature is seen as useful by users [6]. As soon as a new feature is developed, a random sample of users are given access to it. Statistics are then gathered on the interactions users have with the feature and compared to the control group. If the feature utilizes functionality that doesn't exist yet, multiple versions of the feature are built and simultaneously tested against one another.

In the context of this project, A/B testing can be used to integrate any and all UX/UI improvements. For example, to create a dashboard of user and funder statistics, at least three iterations could be drafted and deployed simultaneously. 'User acceptance' could then be a function of tests measuring how long users spent on the dashboard and their actions while on the page.

Chapter 7

Conclusion

The goal of the project was to investigate the feasibility of creating a crowdfunding platform on the Ethereum network to incentivize the development of open source software. Ethereum was selected as it is the first blockchain-based technology that was designed to support extensive scripting in the form of smart contracts. The project aimed to leverage the immutability of the Ethereum blockchain to a more secure and efficient platform for raising bug bounties in open software repositories.

Taking inspiration from the successful open software crowdfunding platform BountySource, the key processes were broken down into three user-stories from which a prototype could be built. The user stories outlined the simple creation of a crowdfund, the funding of a bounty for a particular issue and either the payout of the bounty as a reward to a developer who solved the issue or the cancellation of the crowdfund and returning of all funds back to the funders. The user stories were encoded into smart contracts using the programming language Solidity that has been built to communicate with the Ethereum blockchain.

A front-end user interface was built using the standard Web 2.0 technologies (HTML/CSS/Javascript) and an Ethereum framework called Truffle was employed to link the front-end with an Ethereum blockchain. The first iterations of the platform were built using a local simulation of the Ethereum Network with a blockchain existing only in memory. This allowed for instantaneous transaction times and rapid prototyping. Once the platform was sufficiently tested, it was deployed on the Ethereum Testnet, short for ‘Test Network’. The Ethereum TestNet is a network updating a blockchain with all the same functionality as that of the Mainnet. The benefit of deploying on the Testnet is that ‘Ether’, crypto-currency that pays for computation on the network, is significantly cheaper.

Although two major bugs were found later in the project, the platform managed to satisfy the main user stories as outlined in the Design chapter. Ideally, the platform will be able to fund its own development by posting the bugs as the first crowdfunds in its directory. It is hoped that the work done for this project and the write up will serve as valuable documentation for future development in the Ethereum community.

Bibliography

[1] Bounty source *Bounty Source Statistics*. https://www.bountysource.com/stats. pages 14

[2] Buterin *Ethereum Frontier Guide*. https://ethereum.gitbooks.io/frontier-guide/content/. pages 6, 7

[3] Consensys *A 101 Noob Intro to Programming Smart Contracts on Ethereum*. https://medium.com/@consensys/a-101-noob-intro-to-programming-smart-contracts-on-ethereum-695d15c1dab4#.23abmo1f7. pages 23

[4] Consensys *Smart Contract Best Practices*. https://github.com/consensys/smart-contract-best-practices. pages 24, 25, 27

[5] Consensys *Truffle Documentation*. https://github.com/consensys/truffle. pages 27

[6] Google *Get real user feedback with beta tests*. https://developer.android.com/distribute/engage/beta.html. pages 50

[7] Krawisz *Proof of Work Concept*. http://nakamotoinstitute.org/mempool/the-proof-of-work-concept/#selection-11.8-11.22. pages 6

[8] Reitwiessner *How to Write Safe Smart Contracts*. http://chriseth.github.io/notes/talks/safe_solidity/#/4. pages 23, 24

[9] Reitwiessner *Smart Contract Security - Ethereum Blog*. https://blog.ethereum.org/2016/06/10/smart-contract-security/. pages 23

[10] Unattributed *Half a million widely trusted websites vulnerable to Heartbleed bug*. http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html. pages 13

[11] Unattributed *The Heartbleed Bug*. www.heartbleed.com. pages 13

[12] Username gerv *Should I Start A Non-Profit Organization (NPO) for my FLOSS Project?* http://flossfoundations.org/starting-a-non-profit. pages 12

[13] Andreia Palma (CMS) Amanda Brock (ORI), Roberto Galoppini (GKNT). Floss business models explained. pages 12

[14] Marc Andreessen. Why software is eating the world. Wall Street Journal, August 20 2011. pages 2

[15] David Bretthauer. *Open Software: A History*. U Conn Libraries Published Works (2001). pages 10

[16] V. Buterin et al. Ethereum frontier guide. pages 7, 8, 9

[17] Bill Gates. An open letter to hobbyists. pages 11

[18] Open Software Initiative. Frequently asked questions. pages 10

[19] A. Castillo J. Brito. Bitcoin: A primer for policymakers. pages 6

[20] Niels Jørgensen. Putting it all in the trunk: Incremental software development in the freebsd open source project. *Information Systems Journal*, 11(4):321–336, 2001. pages 16

[21] Sunghun Kim and E. James Whitehead Jr. How long did it take to fix bugs? 2006. pages 13

[22] Steve Marquess. Of money, responsibility, and pride. pages 13

[23] Katherine Noyes. Linux's brilliant 25-year history. PC World, August 25 2016. pages 2

[24] Sarfraz Khurshid Saha, Ripon K. and Dewayne E. Perry. An empirical study of long lived bugs. 2014. pages 13, 14

[25] Richard Stallman. *Free software, free society: Selected essays of Richard M. Stallman*. The Free Software Foundation, 2002. pages 2, 10

[26] Ilka Tuomi. *Learning From Linux*. Meaning Processing. pages 11