

# Final Presentation Master's Thesis: Identification of Programming Patterns in Solidity

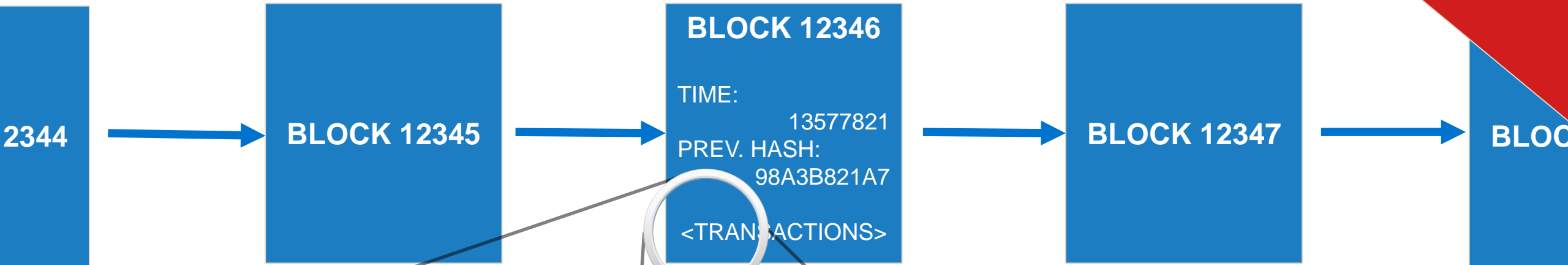
Franz Volland, 04<sup>th</sup> June 2018, Scientific advisor: Ulrich Gellersdörfer


Chair of Software Engineering for Business Information Systems (sebis)  
Faculty of Informatics  
Technische Universität München  
[www.matthes.in.tum.de](http://www.matthes.in.tum.de)


- 1** From Blockchain to Solidity – A Short Introduction
- 2** Motivation
- 3** Research Questions
- 4** Approach
- 5** Pattern Catalog & Categories
- 6** Investigation on Pattern Usage

# From Blockchain to Solidity - A Short Introduction

RECAP



 **bitcoin**  
A → B : 2 XBT  
A → C : 0.2 XBT  
B → C : 1 XBT

 **ethereum**  
A → B : 2 ETH  
A → Code  
B → Code.do ()

**Solidity:** 

- Smart contract programming language
- Similar to JavaScript
- Announced 2014

## Major Hacks:

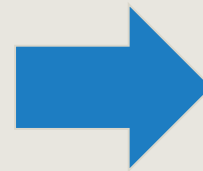
- The DAO: 3.6M  $\text{Ξ}$
- Parity Multisignature Wallet 2x 150k + 514k  $\text{Ξ}$

“We are in Cryptoland. [...] It’s like Australia where anything with a heartbeat will try to kill you.”  
- Martin Swende (Ethereum Foundation)



## Smart contracts are a worthwhile target <sup>1</sup>:

- Finality of transactions
- Monetising successful attacks is straightforward
- Availability of contract source/byte code



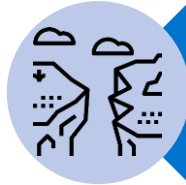
## Patterns (after Alexander<sup>2</sup>):

- Describes a reoccurring problem
- Describes core of the solution
- Reusable solution to a problem

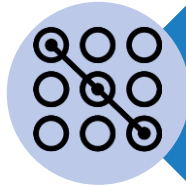
<sup>1</sup> <https://medium.com/@gerhard.wagner/the-phenomena-of-smart-contract-honeypots-755c1f943f7b>

<sup>2</sup> C. Alexander, The Timeless Way of Building. New York: Oxford University Press, 1979

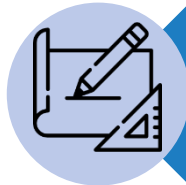
# Research Questions



What are **current challenges** in smart contract development using Solidity?



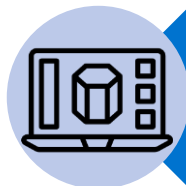
Are there any **best practices or patterns** in smart contract development with Solidity?



How can the identified patterns be **structured and categorized**?



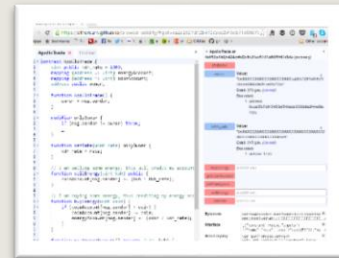
Is there a way to **measure pattern usage** in Ethereum smart contracts?



Are proposed patterns **accepted and implemented** by smart contract developers?

## Grounded Theory approach:

- Papers
- DApp Portals
- GitHub
- Blogs
- Developer Chats
- Code



STATE OF THE DAPPS

||| GITTER



M Medium



# 14 Identified Patterns in 4 Categories

1. Guard Check

2. State Machine

3. Oracle

4. Randomness

Behavioral



5. Access Restriction

6. Check Effects Interactions

7. Secure Ether Transfer

8. Pull over Push

9. Emergency Stop

10. Proxy Delegate

11. Eternal Storage

Security



12. String Equality Comparison

13. Tight Variable Packing

14. Memory Array Building

Upgradeability



Economic



## Modified Gang of Four<sup>3</sup> Taxonomy:

1. Intent

2. Motivation

3. Applicability

4. Participants & Collaborations

5. Implementation

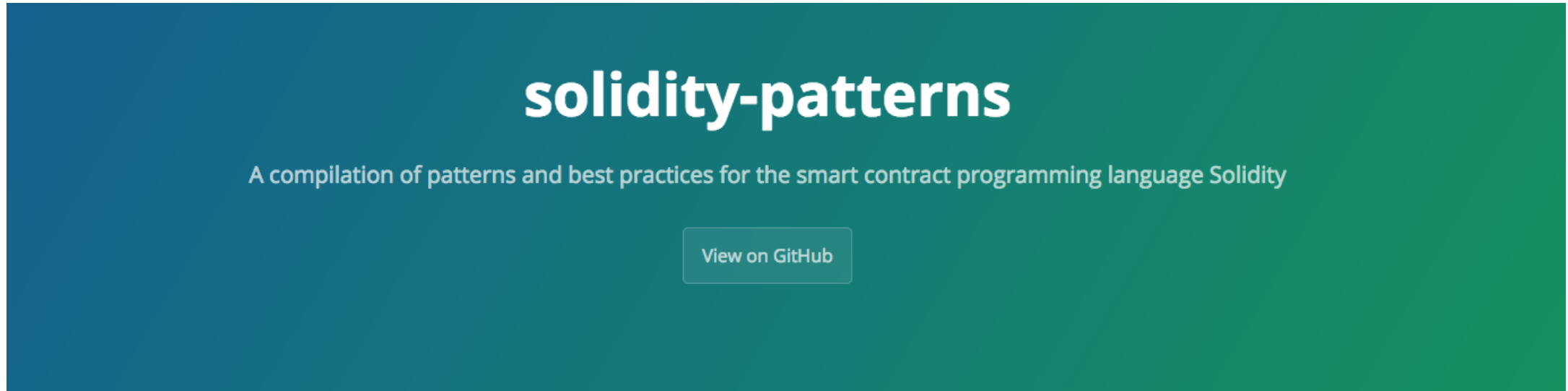
6. Sample Code

7. (Gas Analysis)

8. Consequences

9. Known Uses

<sup>3</sup> Gamma et al.: Design Patterns Elements of Reusable Object-Oriented Software



## State Machine

### Intent

Enable a contract to go through different stages with different corresponding functionality exposed.

### Motivation

Consider a contract that has to transition from an initial state, over several intermediate states, to its final state over his lifetime. At each of the states the contract has to behave in a different way and provide different functionality to its users. The described behavior can be observed in a multitude of use cases: auctions, gambling, crowdfunding, and many more. Even the Solidity documentation



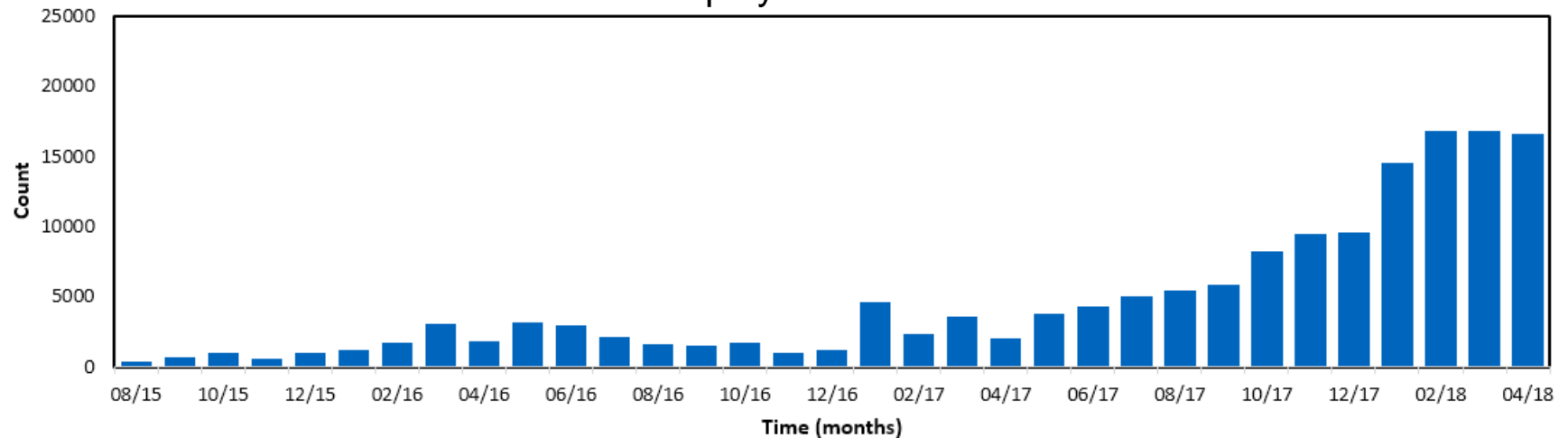
# Investigation of Pattern Usage - Approach

```
for each block in geth.blockchain
    transactions[] = block.getTransactions()
    for each transaction in transactions
        if transaction.to == null
            elasticsearch.post(transaction.code, transaction.timestamp)
```

Pseudocode



Number of distinct deployed smart contracts on Ethereum



# Investigation of Ownable contract with Function Identifiers

**Problem:** Smart contracts are stored in bytecode on the blockchain

Bytecode

```
6060604052341561000f5760003504166306fdde0381146100d1578063095ea7b31461015b57806318160ddd14610191578063
23b872dd146101b6578063313ce567146101de5780633eaaaf86b1461020757806370a082311461021a57806379ba5097f2fde3
8b5780638da5cb5b1461024e57806395d89b411461027d578063a9059cbb10290578063cae9ca51146102b2578063d4ee1d901
4610317578063dc39d06d1461032a578063dd62ed3e1461
```

Solidity Source Code

```
contract Ownable {
  ..
  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }
  ..
}
```

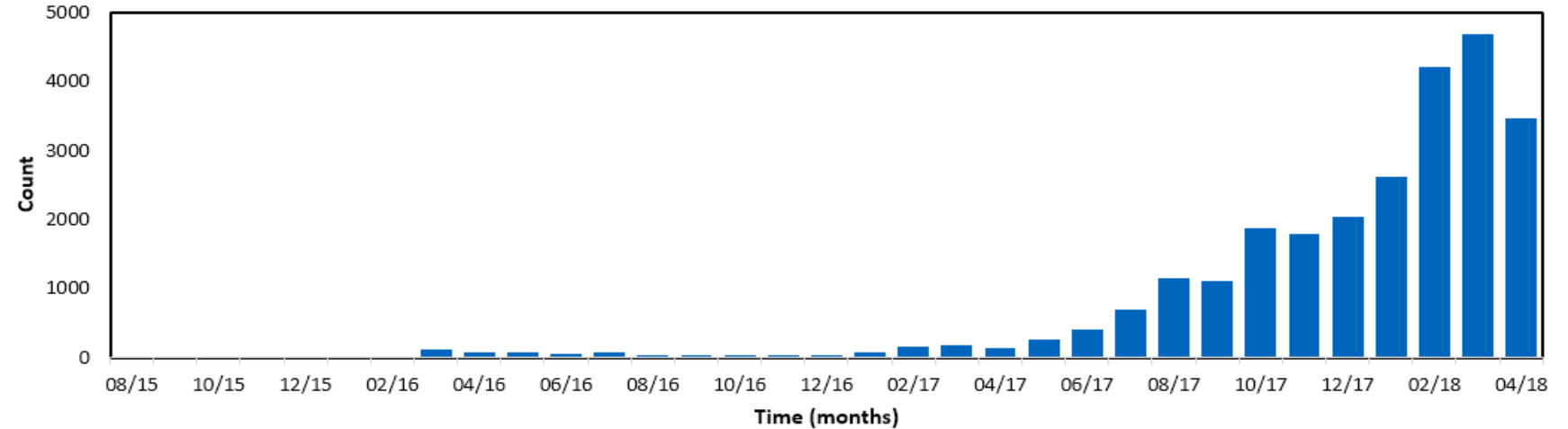
Function Identifier

```
keccak256("transferOwnership(address)") = 0xf2fde38b092330466c661fc723d5289b90272a3e580e3187d1d7ef788506c557
```

# Usage Results of Ownable Contract

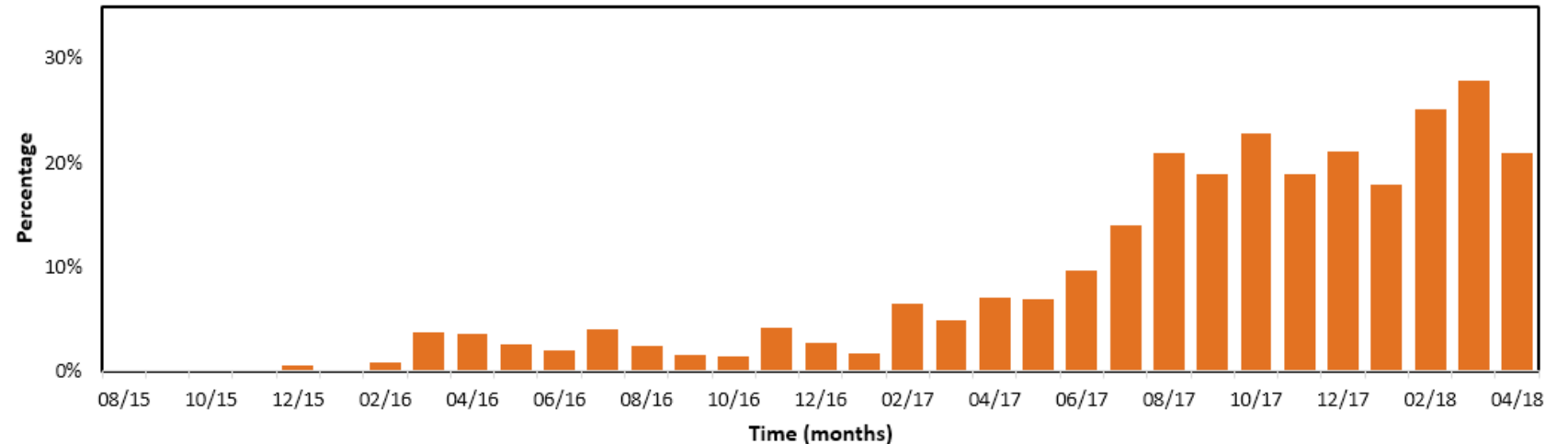
## Absolute occurrences:

- Steady increase in implementations
- Not meaningful on its own

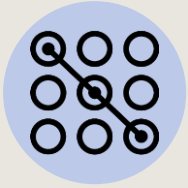


## Relative frequency:

- Increasing usage
- Doubled usage from 10% to over 20% in only two months
- >20% of contracts



## Pattern Catalog



- Developed a structure to present patterns in the context of smart contracts in Solidity
- Identified four categories of patterns
- Identified 14 patterns and assigned them to their respective category



## Pattern Usage Investigation



- Developed a method to investigate pattern usage on contract byte code with the help of function identifiers
- Ownable is used in >20% of distinct deployed smart contracts
- Smart contract developers are in principle willing to use patterns presented to them





## Franz Volland

Technische Universität München  
Faculty of Informatics  
Chair of Software Engineering for Business  
Information Systems

Boltzmannstraße 3  
85748 Garching bei München

Tel +49.89.289.17135

Fra.volland@tum.de  
[www.matthes.in.tum.de](http://www.matthes.in.tum.de)

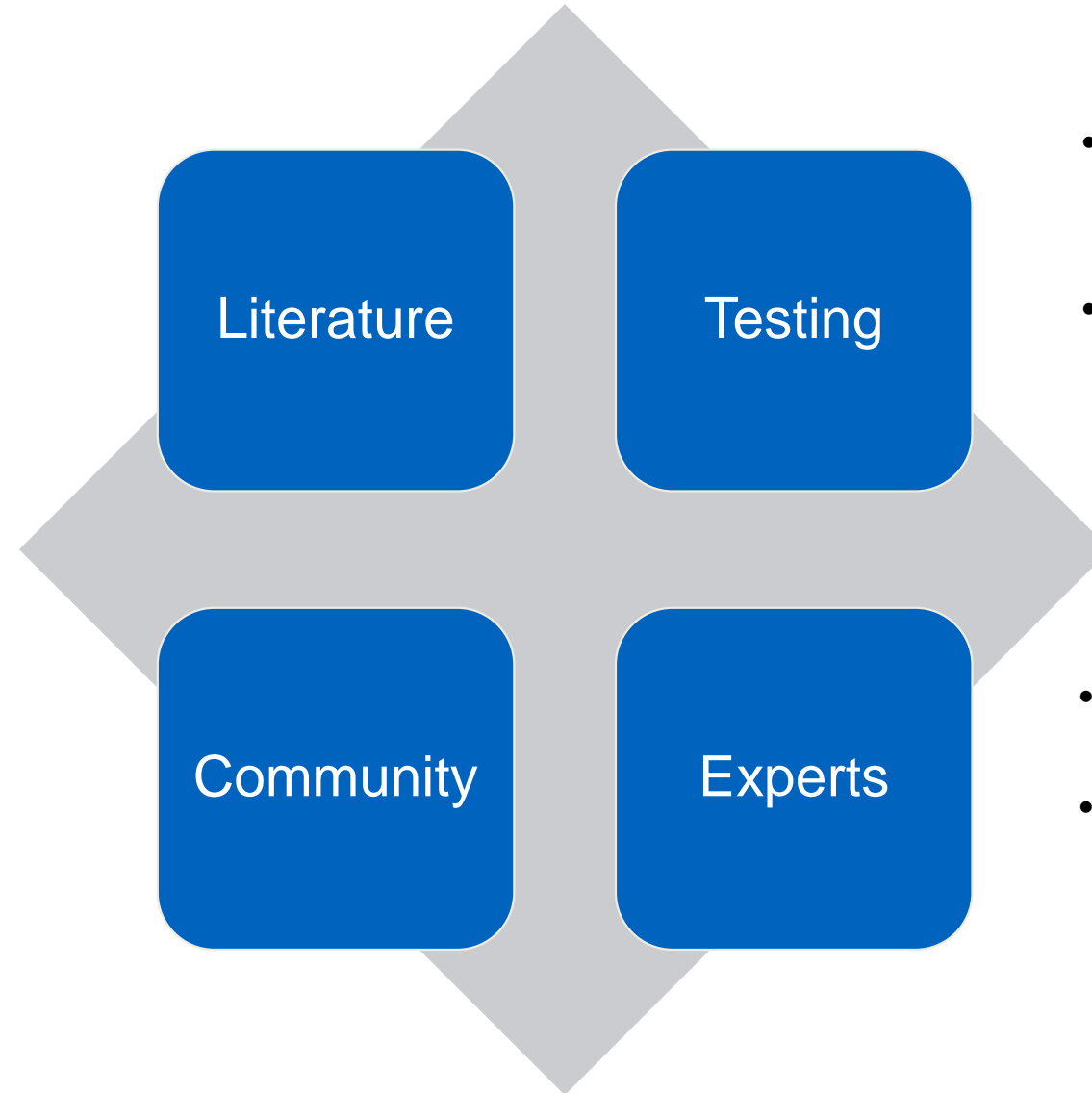
A photograph of a modern, multi-story building with a grey facade and large windows. Three tall flagpoles stand in front of the building, flying the TUM flag (blue with a white geometric pattern), the Finnish flag (white with a blue cross), and the German flag (black, red, and gold horizontal stripes). The building has the words 'MATHEMATIK INFORMATIK' visible on its facade. In the foreground, several people are walking and riding bicycles on a paved area. The sky is clear and blue.

MATHEMATIK INFORMATIK

# Evaluation of Patterns in 4 Parts

- Patterns already validated by original author and peer review

- Patterns were made public and members of developer communities were asked for feedback
- Little success



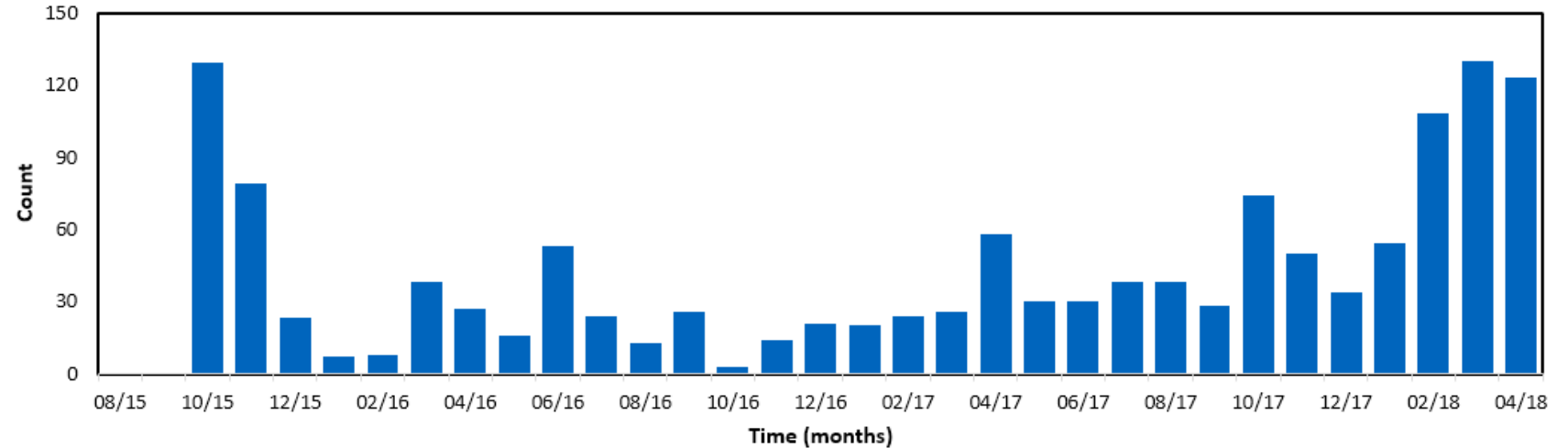
- **Static testing:** code review and with the online compiler Remix
- **Dynamic testing:** Ropsten test net

- Shepherding by a blockchain expert
- Iterative process to improve pattern and eliminate errors

# Usage Results of Oraclize Contract

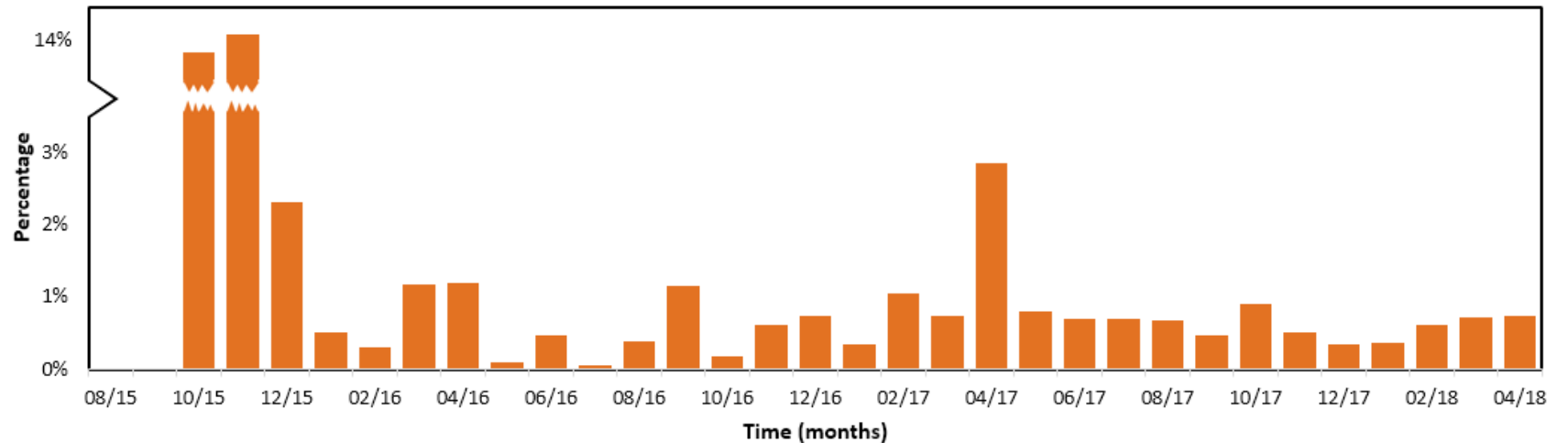
## Absolute occurrences:

- Peaks in the beginning and end of observation period
- Not meaningful on its own



## Relative frequency:

- High usage in the beginning → internal testing
- Usage reaches steady level at little under 1%





This list (original source [here](#)) is as follows:

- The DAO (obviously)
- The “payout index without the underscore” [ponzi](#) (“FirePonzi”)
- The casino with a [public RNG seed](#)
- [Governmental](#) (1100 ETH stuck because payout exceeds gas limit)
- [5800 ETH swiped](#) (by whitehats) from an ETH-backed ERC20 token
- The [King of the Ether game](#)
- Rubixi : Fees stolen because the [constructor function](#) had an incorrect name, allowing [anyone to become the owner](#)
- Rock paper scissors [trivially cheatable](#) because the first to move shows their hand
- Various instances of funds lost because a recipient contained a fallback function that consumed more than 2300 gas, causing sends to them to fail.
- Various instances of call stack limit exceptions.



Vitalik Buterin



#### LATEST POSTS

Roundup Q2  
08th July, 2017

Roundup Round III  
24th May, 2017

# Patterns included in Solidity Documentation

- Solidity by Example
- Solidity in Depth
- Security Considerations
- Using the compiler
- Contract Metadata
- Application Binary Interface Specification
- Joyfully Universal Language for (Inline) Assembly
- Style Guide
- Common Patterns**
  - Withdrawal from Contracts
  - Restricting Access
- State Machine**
  - Example
- List of Known Bugs
- Contributing
- Frequently Asked Questions
- GDPR Compliant Hybrid Cloud: Keep your data in your country with Exoscale.ch
- Read the Docs v: develop

The recommended method of sending funds after an effect is using the withdrawal pattern. Although the most intuitive method of sending Ether, as a result of an effect, is a direct `send` call, this is not recommended as it introduces a potential security risk. You may read more about this on the [Security Considerations](#) page.

This is an example of the withdrawal pattern in practice in a contract where the goal is to send the most money to the contract in order to become the “richest”, inspired by [King of the Ether](#).

In the following contract, if you are usurped as the richest, you will receive the funds of the person who has gone on to become the new richest.

```
pragma solidity ^0.4.11;

contract WithdrawalContract {
    address public richest;
    uint public mostSent;

    mapping (address => uint) pendingWithdrawals;

    function WithdrawalContract() public payable {
        richest = msg.sender;
        mostSent = msg.value;
    }

    function becomeRichest() public payable returns (bool) {
        if (msg.value > mostSent) {
            pendingWithdrawals[richest] += msg.value;
            richest = msg.sender;
            mostSent = msg.value;
            return true;
        } else {
            return false;
        }
    }
}
```

# Programming Language Comparison

Feature	Java	Solidity	Haskell
Programming Paradigm	Object-oriented	Contract-oriented	Functional
Concurrency?	Multi-threading	Serial execution	Multi-threading
Polymorphism?	Through overloading	Through interfaces	Parametric & Ad-hoc
Static/Dynamic Typing?	Statically-typed	Statically-typed	Statically-typed
Strong/Weak Typing?	Strong	Strong	Strong
Higher-order Functions?	With Lambda expressions (Java8)	Not supported	Supported
Inheritance?	Supported	Supported	Not supported
Interfaces?	Supported	Supported	Type classes, similar
Type inference?	With Lambda expressions (Java8)	Supported	Supported
Loops?	Supported	Supported	Not supported
Switches?	Supported	Not supported	Via Case-expression
If-Else?	Supported	Supported	Supported