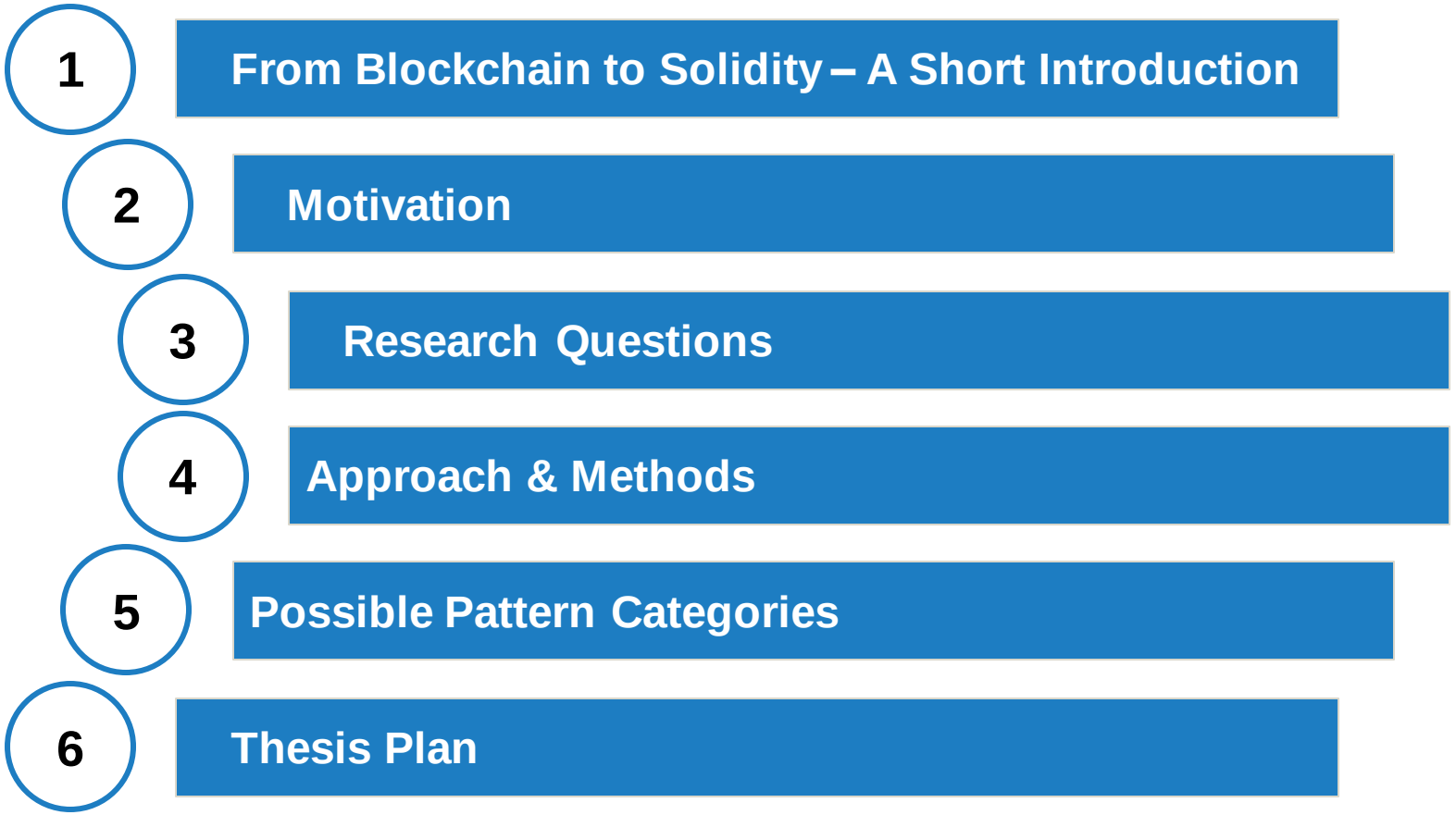
Kickoff Presentation Master's Thesis: Identification of Programming Patterns in Solidity

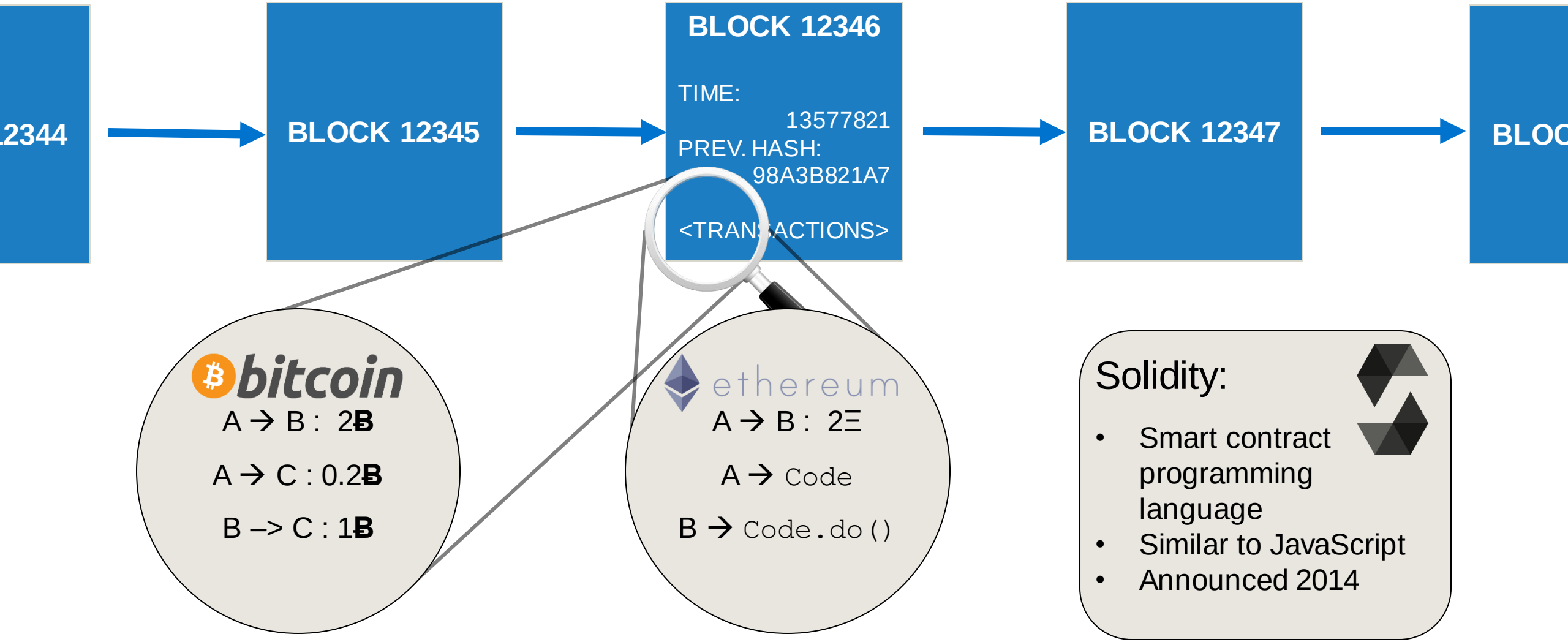
Franz Volland, 29th January 2018, Scientific advisor: Ulrich Gallersdörfer

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

Agenda



From Blockchain to Solidity - A Short Introduction



Solidity:

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

Motivation – Why we need Patterns for Solidity

Major Hacks:

- The DAO: 3.6M Ξ (~3.6 billion $)
- Parity Multisignature Wallet 2x 150k + 514k Ξ (~0.66 billion $)

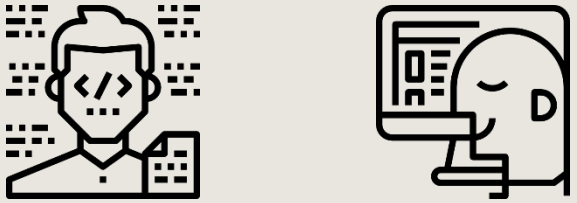
“We are in Cryptoland. […] It‘s like Australia where anything with a heartbeat will try to kill you.“

- Martin Swende (Ethereum Foundation)

Current Problems:

- A lot of attackers
- Language is new for everyone
- Easy to mess up
- High stakes
- Non trivial to understand
- No chance to easily fix mistakes

Solidity Patterns useful for:

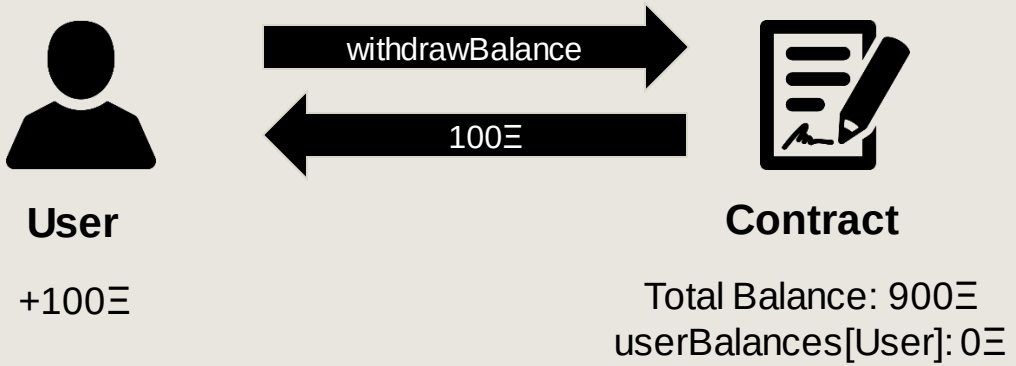


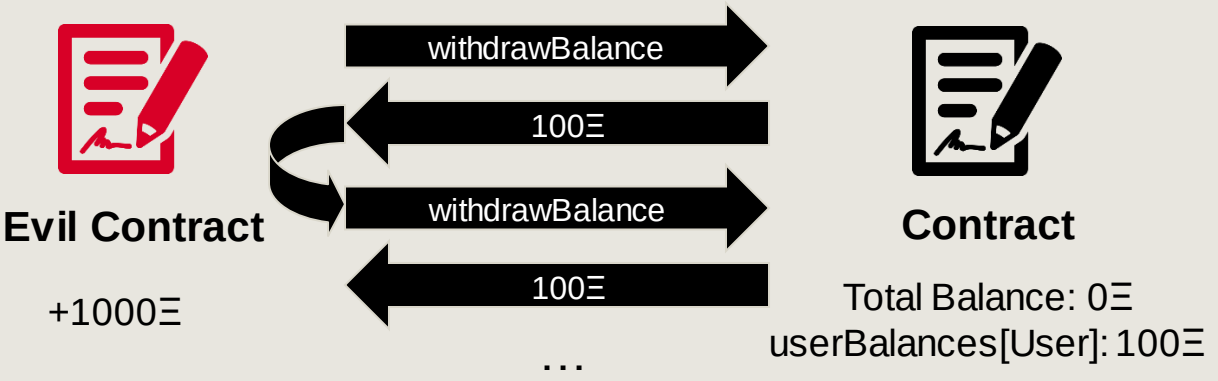
Developers AND **Users**

Motivation – Example Exploit: Reentrancy

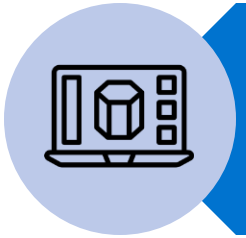
```
1  contract VulnerableAgainstReentrancy {
2
3      mapping (adress => uint) private userBalances;
4
5      function withdrawBalance() public {
6          uint amountToWithdraw = userBalances[msg.sender];
7          if (!(msg.sender.call.value(amountToWithdraw)())) { throw; }
8          userBalances[msg.sender] = 0;
9      }
10
11 }
```

Total Balance: 1000Ξ
userBalances[User]: 100Ξ

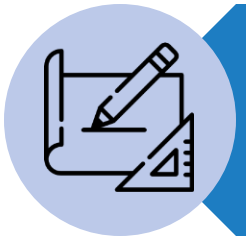




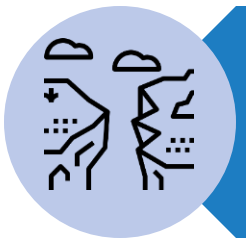
Research Questions



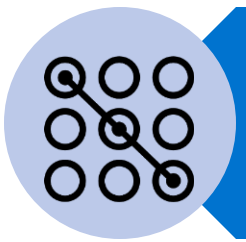
What is the current **state of software engineering** in Solidity?



What is the process of **designing and implementing** smart contracts on the Ethereum blockchain?



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



Are there any **best practices** or **patterns** in smart contract development and how can they be **categorized**?

Approach & Methods

Research on:

- Papers
- DApp Portals
- ICO Portals
- GitHub
- Blogs
- Code

STATE OF THE ĐAPPS

ICO bench

GitHub

Medium

Modified Gang of Four[1] Taxonomy:

1. Intent
2. Also Known As
3. Motivation
4. Applicability
5. Structure
6. Participants
7. Collaboration
8. Consequences
9. Implementation
10. Sample Code
11. Known Uses
12. Related Patterns

1 Gamma et al.: Design Patterns: Elements of Reusable Object-Oriented Software

Possible Pattern Categories

Security

- Access Restriction
- Pull over Push
- Checks-Effects-Interaction
- Secure Transfer

Maintainability

- Upgrading Contracts
- Functionality into Libraries

Utility

- Voting
- Randomness
- Crowdfunding
- Oracle
- Function Scheduling
- Auction
- Bounty

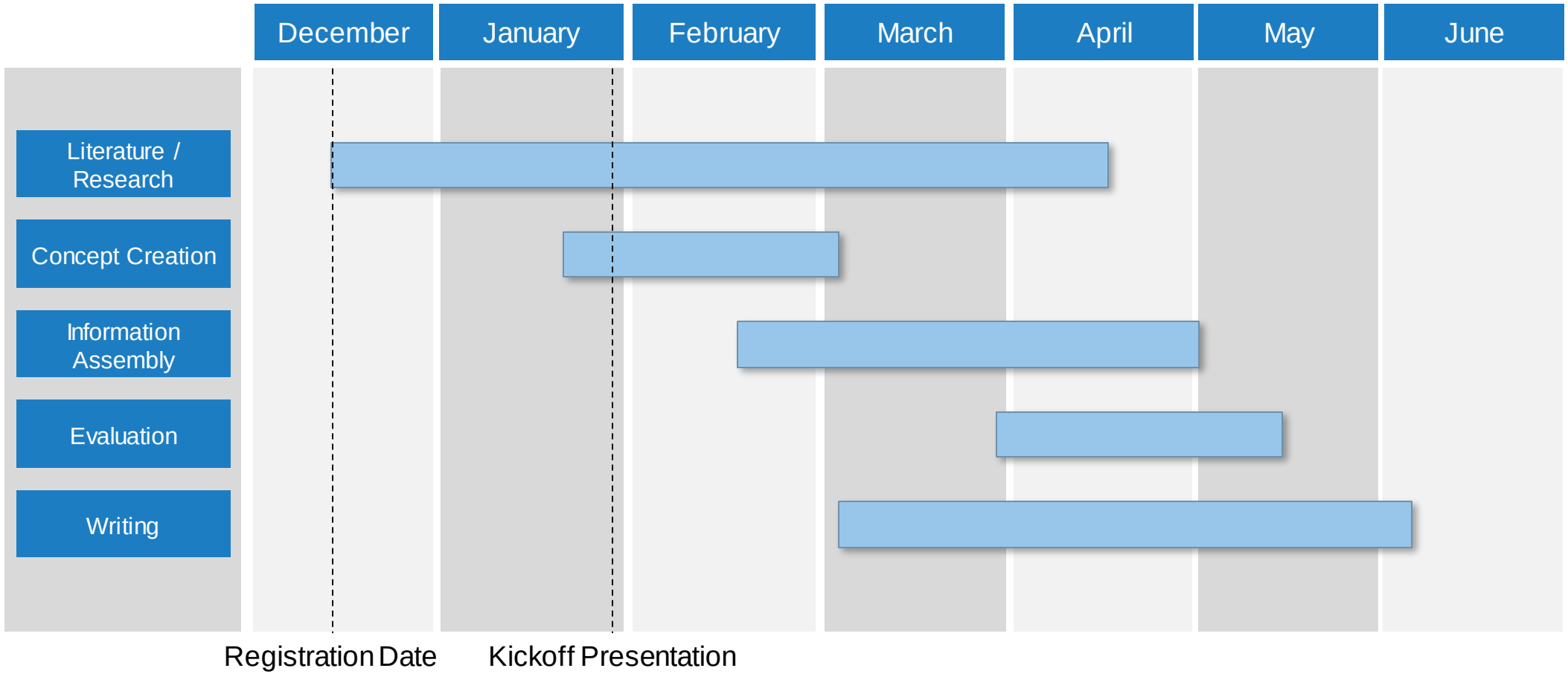
Administration

- State Machine
- String Compare
- Pause
- Assertion
- Suiciding

Economic

- Packing Structs
- Memory Array Building

Thesis Plan



Franz Volland

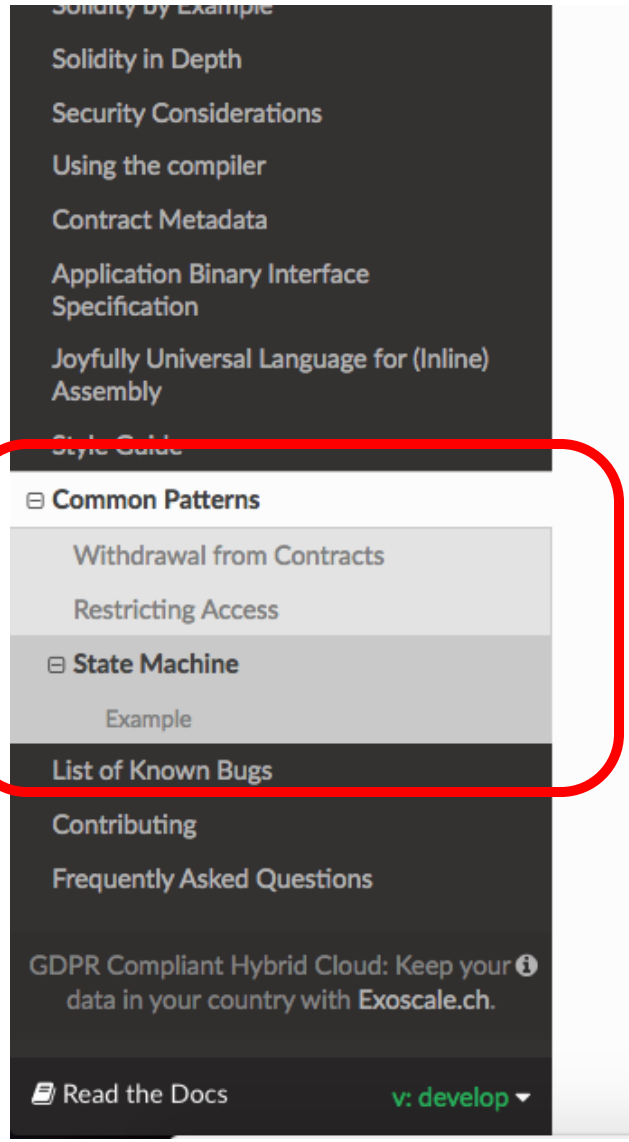
Technische Universität München
Faculty of Informatics
Chair of Software Engineering for Business Information Systems

Boltzmannstraße 3
85748 Garching bei München

Tel +49.89.289.17135

Fra.volland@tum.de
wwwmatthes.in.tum.de

Patterns included in Solidity Documentation



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;
        }
    }
```

Additional Attacks and Blunders

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

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