Scientific School on Blockchain & distributed ledger technology
Pula-Italy 12-15 June 2018

Software engineering methodologies for blockchain development

Michele Marchesi. Roberto Tonelli

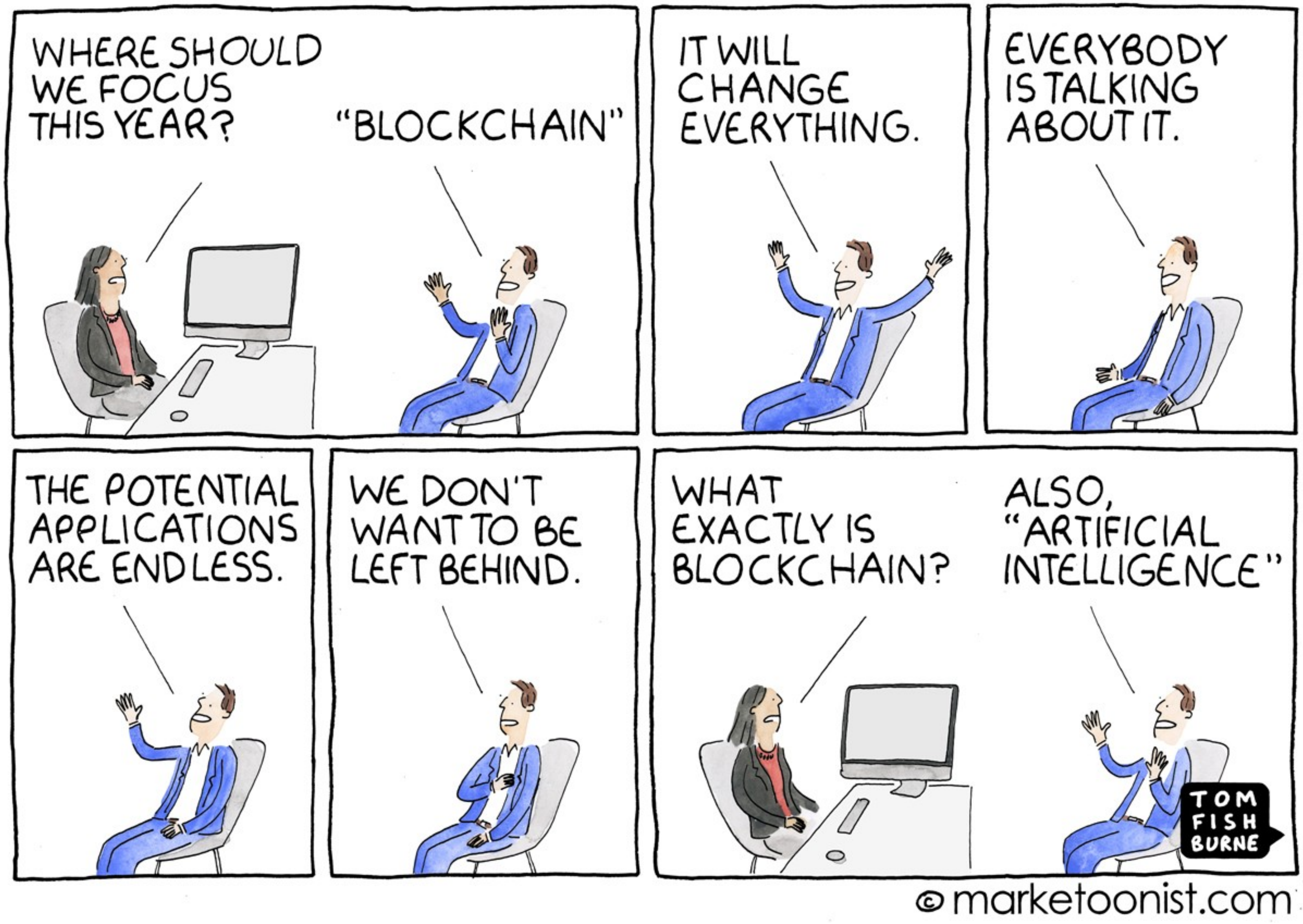
Dept. of Mathematics and Computer Science

University of Cagliari, Italy

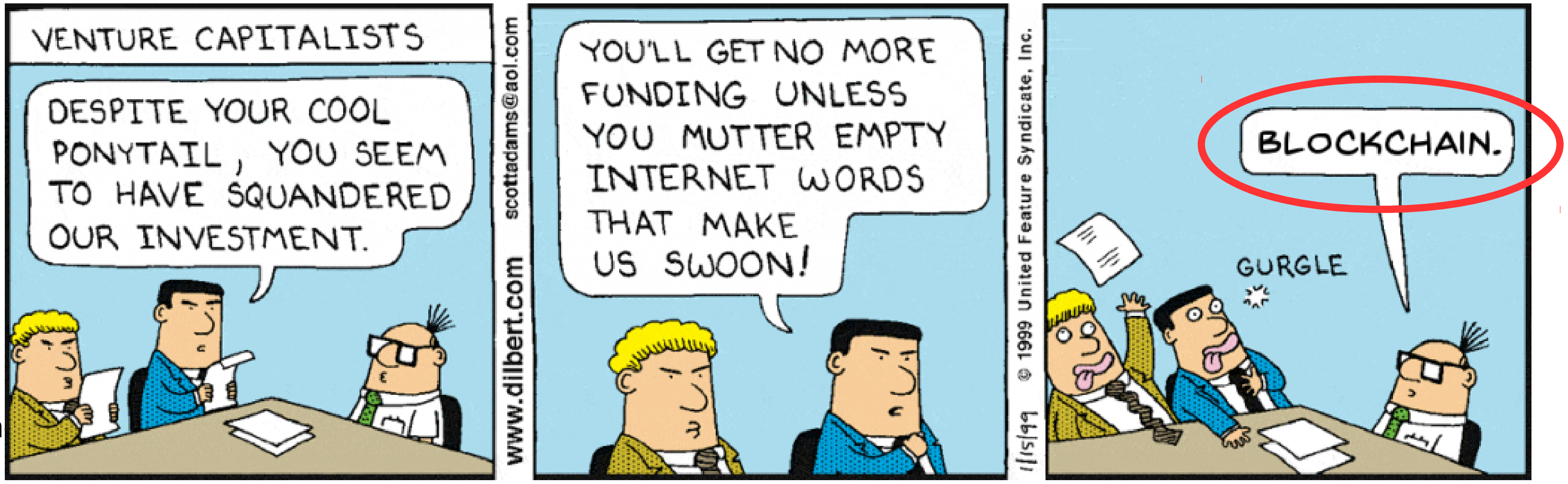
Summary

- IT buzzwords
- How Software Engineering can contribute to blockchain development
- How blockchain development can contribute to Software Engineering
- SCANDALS: a proposal for a process for analysis and design of blockchain applications

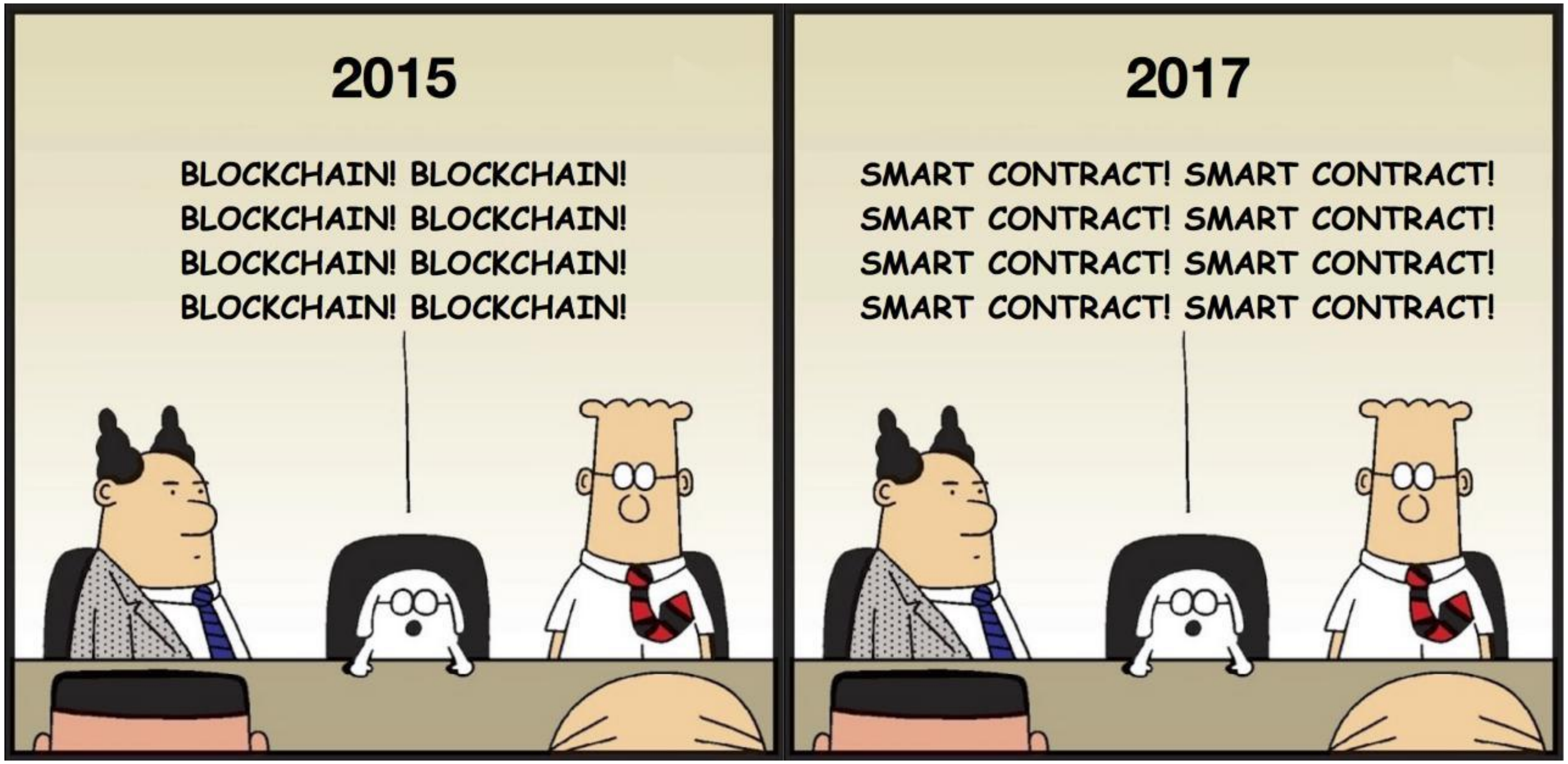
Buzzwords


WHERE SHOULD WE FOCUS THIS YEAR?
"BLOCKCHAIN"
IT WILL CHANGE EVERYTHING.
EVERYBODY IS TALKING ABOUT IT.
THE POTENTIAL APPLICATIONS ARE ENDLESS.
WE DON'T WANT TO BE LEFT BEHIND.
WHAT EXACTLY IS BLOCKCHAIN?
ALSO, "ARTIFICIAL INTELLIGENCE"
TOM FISHBURNE
© marketoonist.com

Recent IT buzzwords

- Cloud computing
- Mobile computing (apps, social networks…)
- Smart cities
- Internet of Things
- Big data
- Green computing
- Artificial Intelligence
- 
VENTURE CAPITALISTS
DESPITE YOUR COOL PONYTAIL, YOU SEEM TO HAVE SQUANDERED OUR INVESTMENT.
scottadams@aol.com
www.dilbert.com
YOU'LL GET NO MORE FUNDING UNLESS YOU MUTTER EMPTY INTERNET WORDS THAT MAKE US SWOON!
© 1999 United Feature Syndicate, Inc.
1/15/99
BLOCKCHAIN.
GURGLE

And also… Smart Contracts!


2015
BLOCKCHAIN! BLOCKCHAIN! BLOCKCHAIN! BLOCKCHAIN! BLOCKCHAIN! BLOCKCHAIN! BLOCKCHAIN! BLOCKCHAIN!
2017
SMART CONTRACT! SMART CONTRACT! SMART CONTRACT! SMART CONTRACT! SMART CONTRACT! SMART CONTRACT! SMART CONTRACT! SMART CONTRACT!

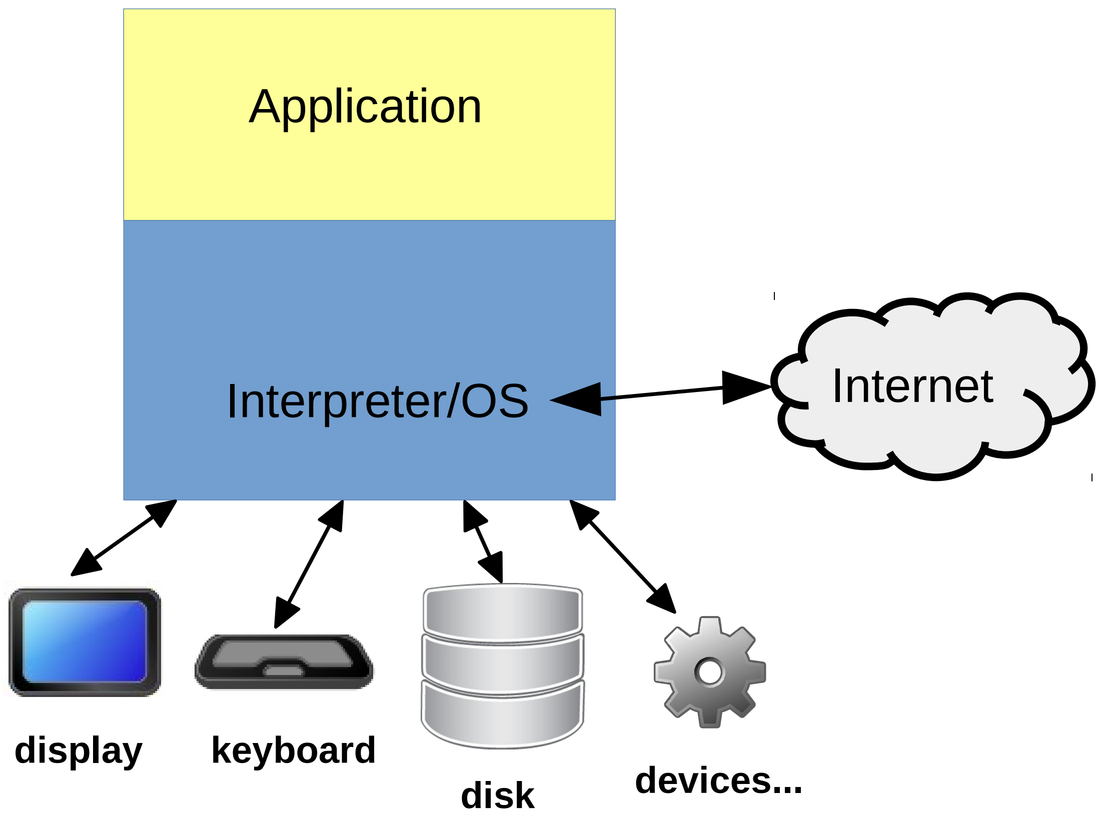
No recent new IT field requires substantially new software development practices . . .

• Cloud computing	distributed computing, scalability
• Mobile computing	programming with reduced resources and some new devices (GPS, accelerometer)
• Smart cities	devices interfacing, data analysis, optimization algorithms
• Internet of Things	devices interfacing, embedded systems
• Big data	databases, data models, scalability
• Green computing	hw/sw interfaces, optimization
• Artificial Intelligence	around since the ‘50s, algorithms
• Blockchain	software running in the nodes: P2P, cryptology, high security requirements

. . . except for Smart Contracts

- Smart Contracts are programs executing on a blockchain
- They run in an isolated environment:
  - The program results must be the same whatever node they run in
  - So, they cannot access the external world (that changes with time)
  - They can only access and send messages to the blockchain itself (that is immutable)
  - Usually, computer programs interact with the external world!
- Moreover, once a SC is deployed on the BC, it is there forever – it cannot be undone or erased!
- Also: strict security requirements, new IDEs, difficulties of testing and deploying, transparency of SCs in the blockchain, ...

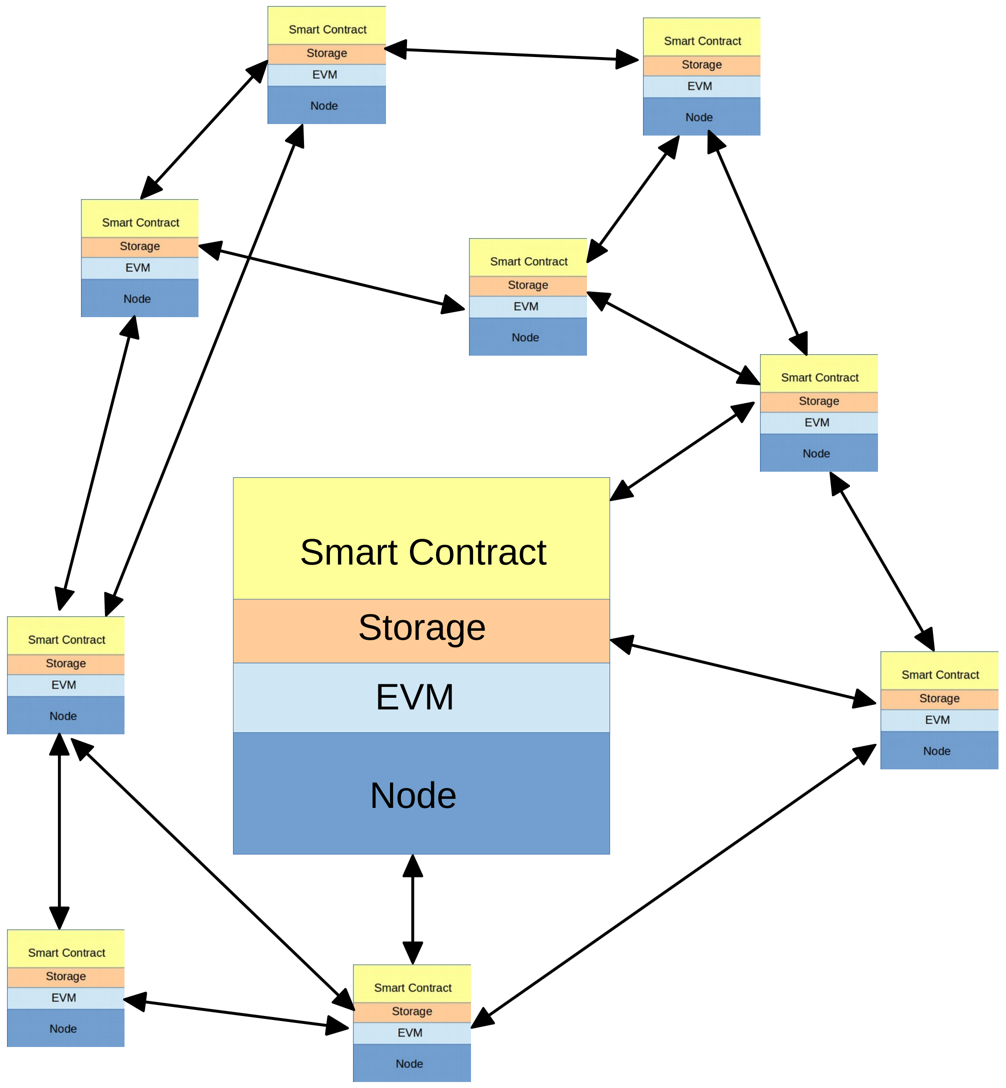
A typical computer application



- The application runs in a computer
- It can access internal and external devices
- It can be activated by any kind of input, including time triggers (via the OS)

A Smart Contract

- The SC runs in every node
- **All executions must produce the same result**
- The calls and the storage modifications are recorded in the Blockchain
- A SC cannot access any device or network



Smart Contracts – the case of Ethereum

- Ethereum SCs are programs residing in the Blockchain
- Written in bytecode interpreted by the EVM
- They are created by special transactions
- They can use other SCs, or inherit from other SCs
- They have a state (kept in Blockchain ***storage variables***)
- Creating a SC and changing its state costs ***GAS*** (Ether)
- A SC is activated (one of more of its functions are called):
  - After its creation (call of its ***constructor***)
  - By a transaction (***message call***)
- A SC can be endowed of Ethers and can send Ethers

Smart Contracts

- A SC, upon a call of one of its functions, can:
  - Create and send a transaction, including_
    - Ether transfer
    - SC creation
    - Sending a message to another SC
  - Call one or more of its functions
  - Change its state (storage variables)
  - Return a value without changing its state or sending a transactions
    - ***In this case, there is no cost***
- A SC **cannot**
  - Initiate an action autonomously (for instance at given times)
  - Access the external world

Software engineering for Blockchain applications

- In the past few years, there has been a strong increase of interest in cryptocurrencies, in Blockchain applications and in Smart Contracts
- This led to a strong inflow of money and of startup ideas
- There are many similarities with the Internet bubble of the second half of the ‘90
- Many projects were born and quickly developed software
- The scenario is that of a **rush to be the first on the market**, fearing of missing out
- This **unruled and hurried** software development does not assure neither software quality, nor that the basic concepts of software engineering are taken into account.

Several epic failures that already happened

- In 2014, the largest Bitcoin Exchange, MtGox, was hacked 850.000 Bitcoins were stolen, valued about $450M
- In 2015, Bitstamp exchange was hacked and 10,000 BTC were stolen (worth $5M)
- In 2016, Bitfinex exchange was hacked and 120,000 BTC were stolen (worth $72M)
- In 2017, Coincheck exchange was hacked and 523 million NEM tokens were stolen (worth $420M)
- In 2016, the DAO smart contract on Ethereum was exploited due to a bug, and the hacker stole 3.6 million ETH, for a value of about $50M
- In 2017, the Parity wallet SC was erroneously disabled, leading to a loss of $2-300M, maybe forever

Two kinds of Blockchain software development

- Development of software
  - to manage a blockchain (running on a node)
  - to implement applications using a blockchain:
    - Wallets
    - Notarization
    - Exchanges
    - DApps . . .
- Development of Smart Contracts
  - ***with very different issues compared with traditional software development***

Both kinds of developments poses significant challenges to software engineering

- Need of new professional roles
- Emphasis on specific security and testing practices
- Specific design notations, macroarchitecture patterns, and meta-models
- Specific modeling languages
- Need of Metrics
- Need of Tools

New professional roles

- The Blockchain is not just software – it has also a strong business nature
- Finance and legal subjects have shown increasing interest toward Blockchain software
- The Blockchain sector will need professional figures with a well-defined skills portfolio which includes finance, law, and technology expertise
- An example of a new role could be an intermediary between business-focused contractors with low technology expertise and IT professionals, aimed to define and test SCs

Specific security and testing practices

- The main problem: once a Smart Contract has been uploaded to the Blockchain, there is no way to change or stop it. **A Blockchain registration is forever!**
  - ...unless the Smart Contract was designed to be stopped (Emergency Stop) or “suicided”
  - but even in this case, the utmost care must be devoted to manage all possible outcomes!
  - because usually the Smart Contract, being a contract, has financial and legal implications

Specific security and testing practices

- The Blockchain guarantees data integrity and uniqueness – it can be trusted!
- Unfortunately, the client software built around a Blockchain is not proven trusted the same way
- Ensuring security and reliability in BOS development might require specific methodologies (might be Cleanroom Software Engineering or thorough software reviews...)
- Furthermore, mathematically sound analysis techniques could help enforcing reliability and security-related properties in BO applications

Specific security and testing practices

- Testing techniques can also enhance system security and reliability
- IBM recently expressed the need for continuous testing techniques to ensure blockchain software quality
- Some testing frameworks are already available for SCs:
  - Ethereum Tester
  - Ganache
  - Populus

Specific design notations, macroarchitecture patterns, and meta-models

- Software engineers should define criteria for:
  - selecting the most appropriate blockchain implementation, if any
  - evaluating the adoption of sidechain technology, or the implementation of an ad-hoc blockchain
  - designing the architecture of a Blockchain system

Modeling languages

- Blockchain-oriented systems may require specialized graphic models for representation
- More specifically, existing models might also be adapted to BOS
- UML diagrams may be modified or created anew to account for the BOS specificities
- For example, diagrams such as the Use Case Diagram, Activity Diagram, and State Diagram could not effectively represent the BOS environment.

Metrics

- BOSE may benefit from the introduction of specific metrics
- A BOS is intrinsically distributed software, specific metrics are required to measure complexity, communication, performance of such systems
- New metrics, both static and dynamic, should take into account the quality of the software
- But also, for example
  - resource consumption (e.g. the so-called gas in the ethereum system)
  - modifiers, events, ...

Also – Blockchain possible uses in software engineering

- The blockchain is a **source of trust**
- It can be used to record information that is
  - transparent
  - immutable
- Smart Contracts can be executed on it, enabling checks and payments
- A software development process might be controlled through SCs, triggering payments to developers
- The software itself might be linked to a blockchain, enabling pay-per-use in a transparent and censorship-resistant way

Notarizing empirical data

- In software engineering, as in many other empirical sciences, the truthfulness and immutability of data is often a problem
- Empirical data used in research (for instance, source code, bug tracking issues, user stories, etc.) can be secured using hash signatures, stored in a blockchain
- It is even possible to encrypt these data, and allow access only to authorized people, through Smart Contracts
- In this way, data integrity can be guaranteed, and experiments can be performed by different research teams on the same data

Smart Contracts for Agile Development

- Goal of the system:
  - To manage the acceptance phase, and related payments to the developers, of an agile project, performed using Scrum process
- The project development is divided in User Stories implementation (or features, or MMFs)
- The implementation flow is in Sprints (groupings of US to implement in a given time interval)
- Each US is provided of one or more Acceptance Tests
  - AT's correct result is coded in a Json file, whose hash digest is registered in the blockchain

Smart Contracts for Agile Development

- When a US is correctly implemented by the developer who committed to it, the ATs associated to the US should pass, generating Json files identical to the correct ones
- A Smart Contract is activated, that verifies that the hash digest of the prescribed AT outputs are the same of those passed.
- This can trigger the payment to the developer, in Ether or in tokens convertible in fiat money (US$).
- Many variations involving the customer, the Product Owner and the developers can be made.
- Each actor is identified through his/her unique private key

Consequently: Call for action!

- Blockchain technology is all about ***trust***
- Blockchain development witnessed an explosive increase in past few years
- There is a rush to write software, like in the Internet rush of the '90s
- Software Engineering practices could greatly improve the quality and reliability of Blockchain software, increasing the ***trust*** on the whole approach
- SE community just started to be aware of the issue, but I forecast an explosive growth in the next few years
- There are plenty of SE fields (only slighly touched in this presentation) whose applications to Blockchain development could be really useful
- **If you start now, you'll be among the first movers!**

Part II - SCANDALS

- Smart Contract ANalysis and Design Agile and Lean System → (Agile and Lean System for Smart Contract Analysis and Design)
- A first attempt to devise a systematic approach to the requirement analysis, design, implementation and testing of a Dapp
- It is based on Agile and Lean practices
  - Agile practices: proven practices enabling to develop software in the presence of changing requirements
  - Lean development: an approach focused on the maximization of the value given to the customer
- … but also to other approaches

The case for Agile

- Agile methods are suited to develop system whose requirements are not completely understood, or tend to change. These characteristics are present in DApps:
  - DApps are typically very innovative applications
  - Often, there is a run to write a DApp to be the first who launches it on the market
- Agile is suited for small, self-organizing teams working together, as is the case for many DApp teams
- The customer or the Product Owner (expert in the system requirements) is highly available to the team
- Agile is iterative and incremental with short iterations, and is suited to deliver quickly and to deliver often – this would be appreciated in the context of DApp development

The case for Agile (2)

- Other practices of Agile development that are very well suited to DApp development are:
  - Continuous Testing, Test Driven Design
  - Refactoring
  - Continuous Integration
  - Collective code ownership
  - Information Radiators (Cards, Boards, Burndown charts)
  - Coding Standards
  - Pair Programming (in some cases)
- On the other hand, DApps have very strict security requirements, and a more formal approach with respect to some aspects of the development could be useful

The case for Lean

- Lean software development is an approach that maximizes the value given to the customer
- Lean principles:
  - Add nothing but value (Eliminate Waste)
  - Center on the people who add value
  - Flow value from demand
  - Optimize across organizations
- In practice, Agile and Lean complement each other
- The use of a Kanban board to drive development is an option – in this case, iterations are dropped, and a continuous flow of incremental features is delivered

The case for more traditional design

- Key factors in SC design:
  - Data: permanent data are very expensive, so they must be thoroughly designed
  - Interactions: they are key to system proper behavior, and the source of all attacks
  - Security: if there is a possible exploit, **it will be exploited!** Security patterns, code inspection and detailed tests must be applied to get a reasonable security level
  - Documentation: in some cases, documentation in the code is the best solution. In other cases, better to keep the code obscured and the documentation separated from the code

The case for more traditional design (2)

- Possible tools:
  - Data: UML class diagrams with proper stereotypes to represent SC concepts; UML state diagrams when needed.
  - Interactions:UML sequence diagrams with proper extensions to represent all possible kinds of interactions, including transfer of money
  - Security: checklist of patterns to control, code inspections, possible automated tools for code verification and testing
  - Documentation: UML diagrammer, documentation standards
  - Metrics: compute standard and specific metrics on the code and on the SC usage (inspecting the blockchain)

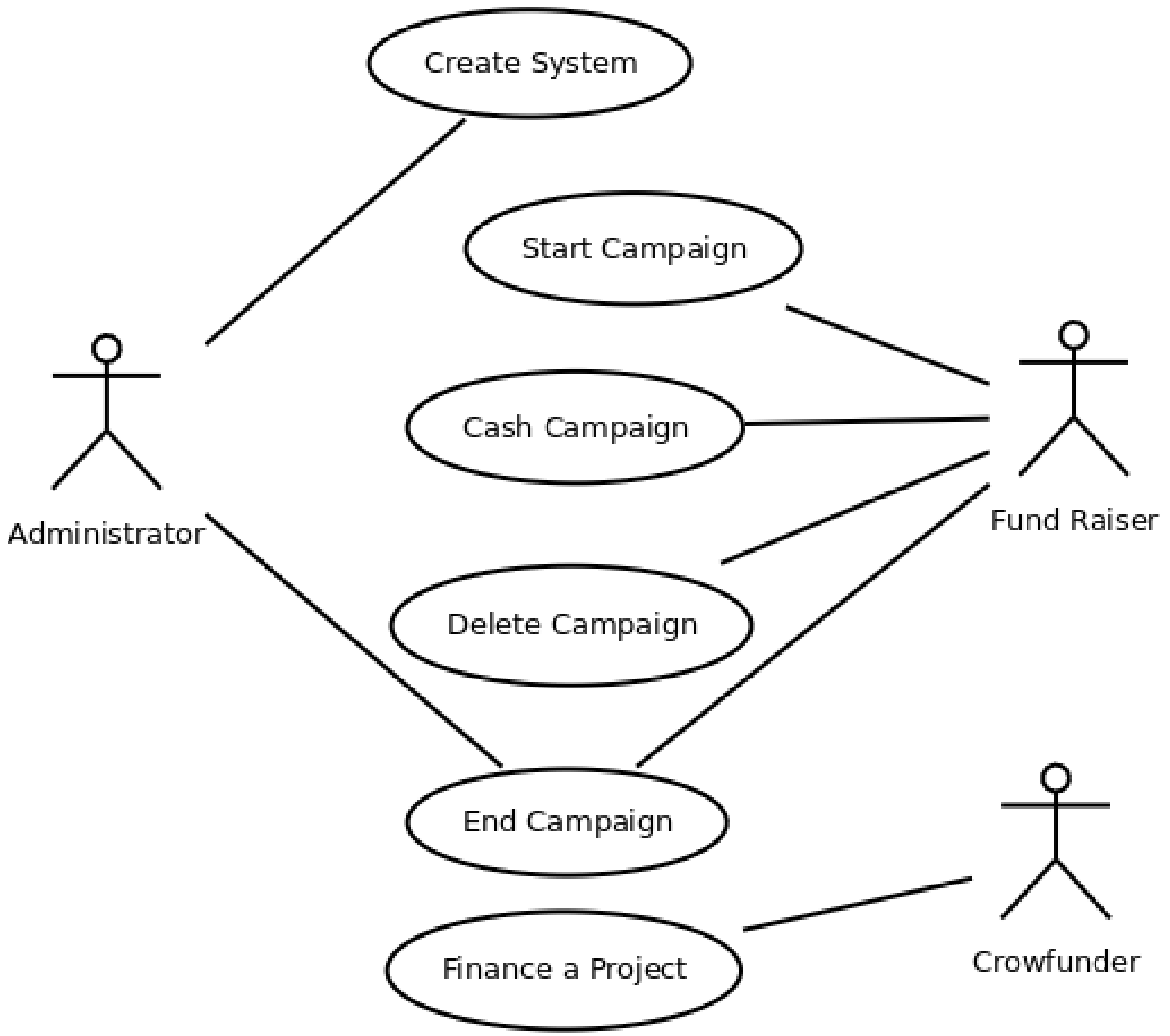
SCANDALS Steps (preliminary)

1. **Define in one or two sentences the goal of the system. For instance:** *To create a simple crowfunding system, managing various projects that can be financed using Ethers*
2. **Identify the actors (human roles, external systems and devices). For instance:**
    1. *System Administrator: he accepts the projects and their property; takes action in the case of problems*
    2. *Fund Raiser: they give the crowfunding project data, including the address receiving the money*
    3. *Crowfunder: they finance projects sending Ethers*

Step 3 – User Stories

- Define the User Stories of the system.
  - *Create System: The Administrator creates the contract, that register his address*
  - *Start Campaign: A Fund Raiser activates a CF project, giving its data: soft and hard cap, end date, address where to send money to*
  - *Cash Campaign: The Fund Raiser, if the time of the CF has expired, or if the hard cap has been reached, cashes out the Ethers given to the project*
  - *Delete Campaign: The Fund Raiser cancels the project; the Ethers are given back to Crowfunders*
  - *End Campaign: The Administrator, or the Fund Raiser, if the time of the CF has expired and the soft cap has not been reached, ends the project; the Ethers are given back to Crowfunders*
  - *Finance a Project: a Crowfunders sends Ethers to a project*

Step 3 – UML Use Case (User Stories) Diagram



Step 4 – Divide into SC system and external system

- Divide the system into two separate systems:
  - The system operating on the blockchain, composed of SCs
  - The system that interacts with the formes, consisting of clients (and possibly servers)
- The SC system interacts with the outside exclusively through blockchain transactions.
  - It has actors, recognized by the respective address
  - It can use libraries and external contracts
  - It can generate transactions to other contracts, or can send Ethers
- The client / server system is the one described in the previous steps
  - But it adds the interface to the SCs

Step 5A – Design of the Smart Contract(s)

- Identify the external actors (recognized by the address)
- Define the decomposition in Smart Contracts:
  - Only one SC
  - More interacting SCs
- For each of the SC identified:
  - Define used libraries and external contracts
  - Define the relevant actors
  - Define the other SCs from which it receives and to which it sends transactions or Ethers
  - If necessary, define a state diagram
  - Define the data structure
  - Define the external interface (ABI) and the events
  - Define internal functions and modifiers
  - Define the tests and the security assessment practices

Step 5B – Design of the external system

- Redefine the actors and the user stories, adding the new (passive) actors represented by the SCs
- Define the acceptance tests of the subsystem
- Decide the broad architecture of the system
- Define the User Interface of the relevant modules
- Perform an analysis of the system, defining:
  - The decomposition in modules
  - The structure and storage of permanent data
  - The data or class structure of the application(s)
  - The connections and the flow of messages between participants, including the SCs
  - The state diagrams (if needed)
  - The detailed interfaces of the various modules .
  - The response to the events raised by SCs

SC modeling – key tool is UML class diagram

- Model as classes (with proper stereotypes):
  - contracts
  - interfaces
  - libraries
  - structs
  - enums
- Model data location (storage/memory)
- Model collections (map, array)
- Model hash digests related to data
- Use patterns and anti-patterns
- Use UML state diagrams for relevant contracts
- Include a State-Function table to show which functions are callable in which state(s)

Dynamic SC modeling

- Use UML Sequence Diagrams to model messaging:
  - different kinds of participants (identified by their accounts):
    - SCs, including Oracles
    - Ether Accounts
    - Persons
    - External systems
  - different kinds of messages:
    - SC creation
    - function call
    - view/pure function call
    - ETH transfers

Security Analysis

- Security Analysis services:
  - https://tool.smartdec.net/
  - Use style controls of Remix
  - https://securify.ch/
- Apply security patterns/antipatterns
- Use Code reviews
- Use of secure math operations
- Check if Complexity metrics is too high

A Case Study: Corporate assembly management

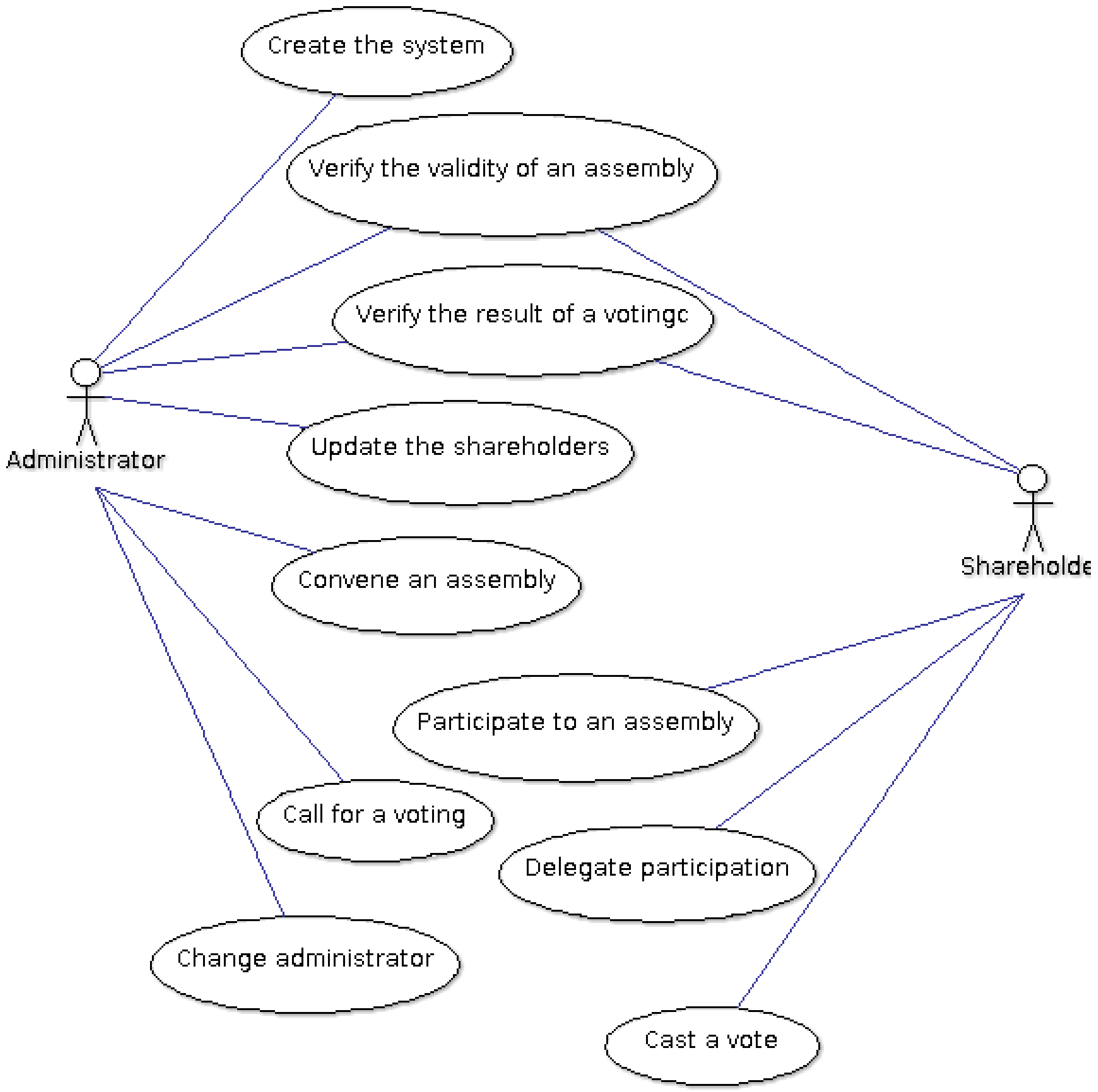
1. Goal of the system:

- To manage in a simplified way voting in corporate assemblies

2. Actors:

- **Corporate administrator**: manages the system, manages the shareholders and their shares, convenes assemblies, calls for votings.
- **Shareholder**: participates to assemblies, casts his votes, delegates participation to assemblies.

3. User stories:



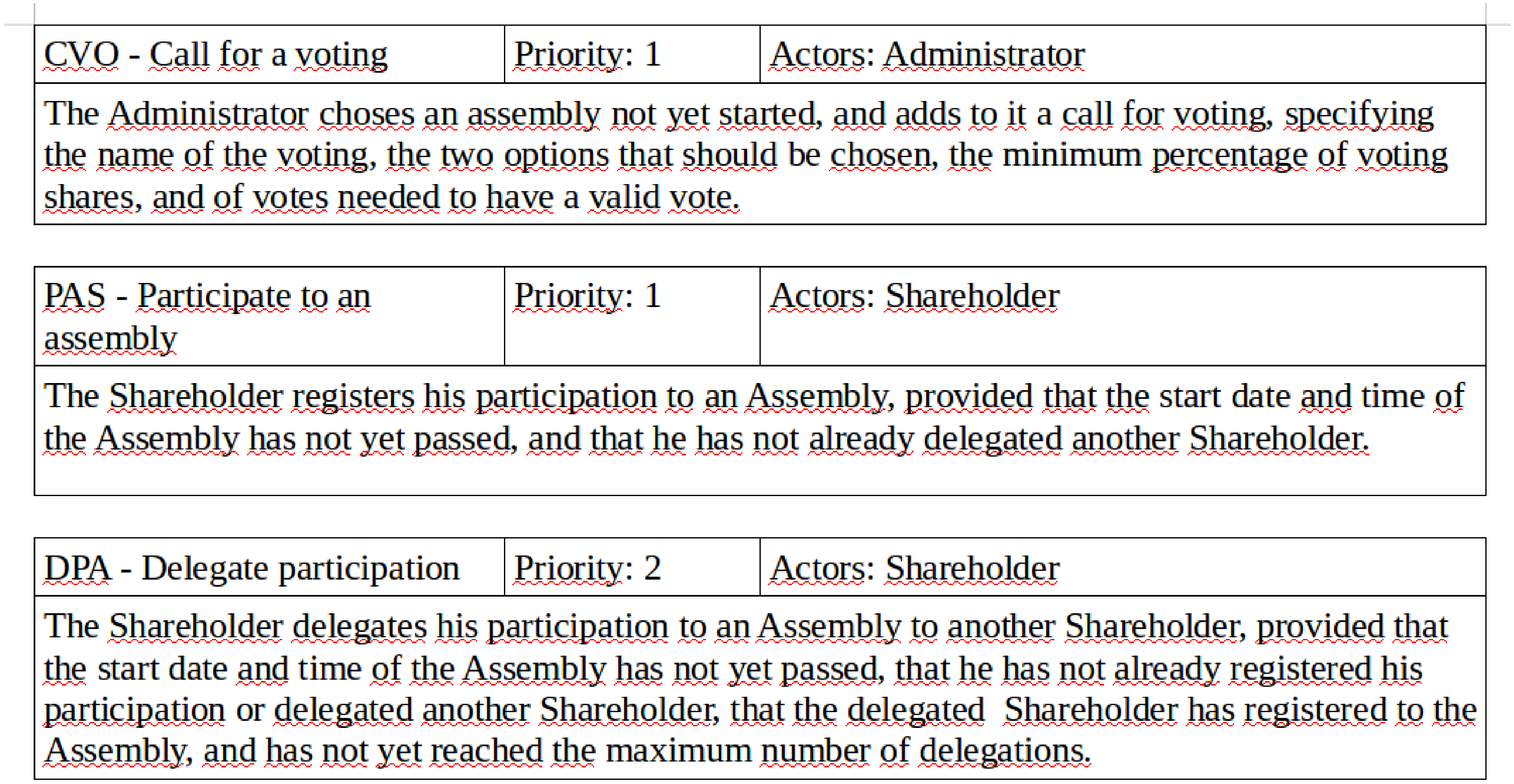
3. User stories:

CRS - Create the system	Priority: 1	Actors: Administrator
The Administrator creates the system related to a firm, inputting the name of the firm, the Administrator’s name and, for each shareholder: name, address and number of shares. The Administrator can be one of the shareholders.		
UPS - Update the shareholders and their shares	Priority: 2	Actors: Administrator
The Administrator modifies the data pertaining to a shareholder. He can delete the shareholder, modify his shares, or add a new shareholder, giving name, address and number of shares.		

CHA - Change administrator	Priority: 3	Actors: Administrator
The Administrator nominates a new Administrator, giving the new name and the new Administrator’s address.		

CAS - Convene an assembly	Priority: 1	Actors: Administrator
The Administrator add an assembly related to a firm, inputting the start and end date and time of the assembly, a short description of its goal(s), the minimum percentage of shares needed for its validity, and the maximum number of delegations that can be given to a single Shareholder. No existing assembly can overlap with the new one, so there is at most one active assembly (or no active assembly at all).		

3. User stories:



CVO - Call for a voting	Priority: 1	Actors: Administrator
The Administrator choses an assembly not yet started, and adds to it a call for voting, specifying the name of the voting, the two options that should be chosen, the minimum percentage of voting shares, and of votes needed to have a valid vote.		

PAS - Participate to an assembly	Priority: 1	Actors: Shareholder
The Shareholder registers his participation to an Assembly, provided that the start date and time of the Assembly has not yet passed, and that he has not already delegated another Shareholder.		

DPA - Delegate participation	Priority: 2	Actors: Shareholder
The Shareholder delegates his participation to an Assembly to another Shareholder, provided that the start date and time of the Assembly has not yet passed, that he has not already registered his participation or delegated another Shareholder, that the delegated Shareholder has registered to the Assembly, and has not yet reached the maximum number of delegations.		

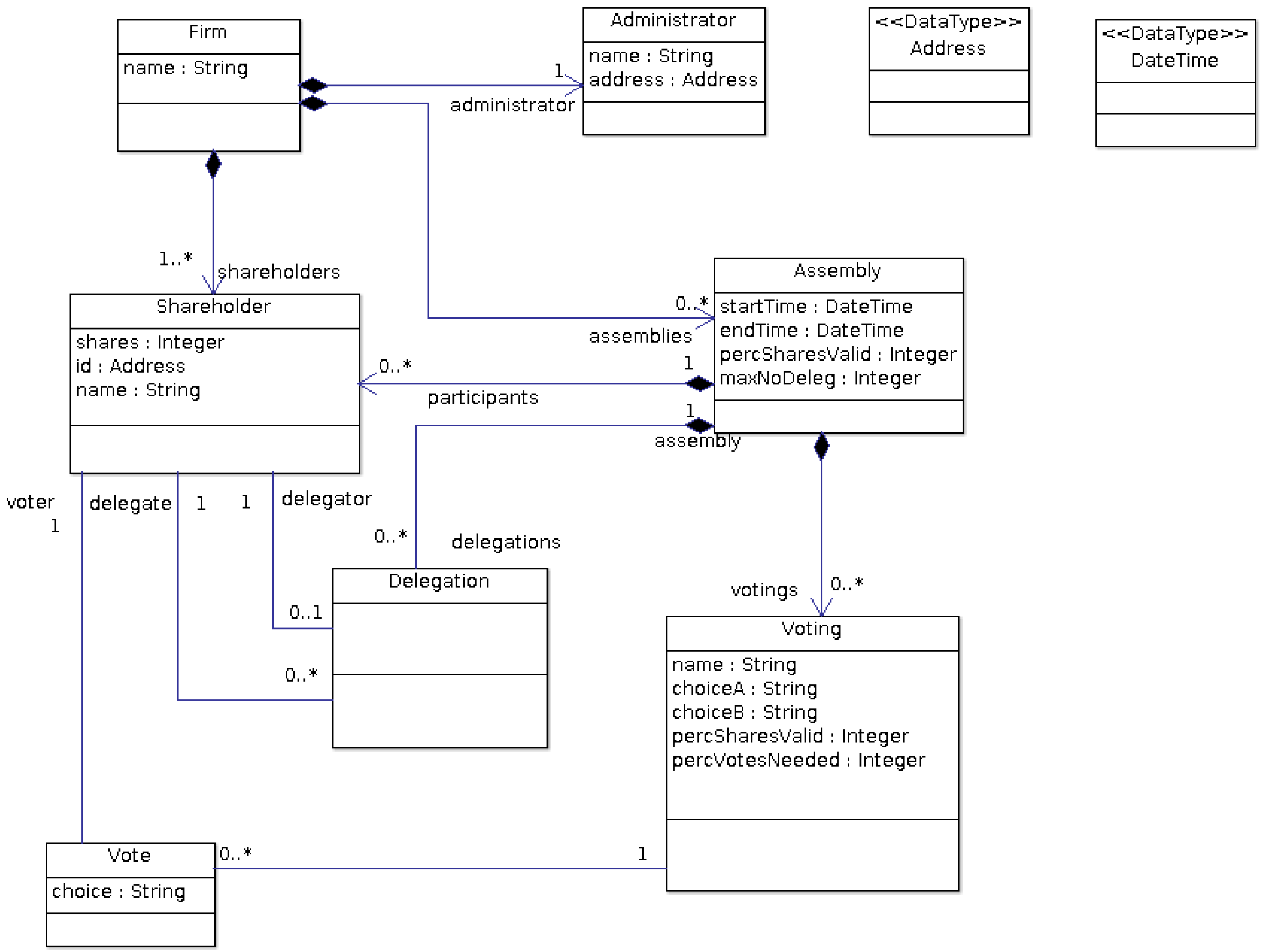
3. User stories:

CVT - Cast a vote	Priority: 1	Actors: Shareholder
The Shareholder casts his vote for one of the choices of a given voting, provided that he is participating to the Assembly of the voting, that this Assembly has started and has not yet expired, and that he has not delegated another Shareholder.		

VVA - Verify the validity of an assembly	Priority: 2	Actors: Shareholder, Administrator
The Shareholder or Administrator, given the date of an Assembly, reads the total number of shares that participated to the Assembly, and checks if the minimum number has been reached.		

VRV - Verify the result of a voting	Priority: 2	Actors: Shareholder, Administrator
The Shareholder or Administrator, given the date of an Assembly, and the name of a voting, reads the voting results (choice 1, choice 2 or no choice).		

Step 3 – The data structure representing this system shown using a UML class diagram.



Step 4 – Divide the system into two subsystems

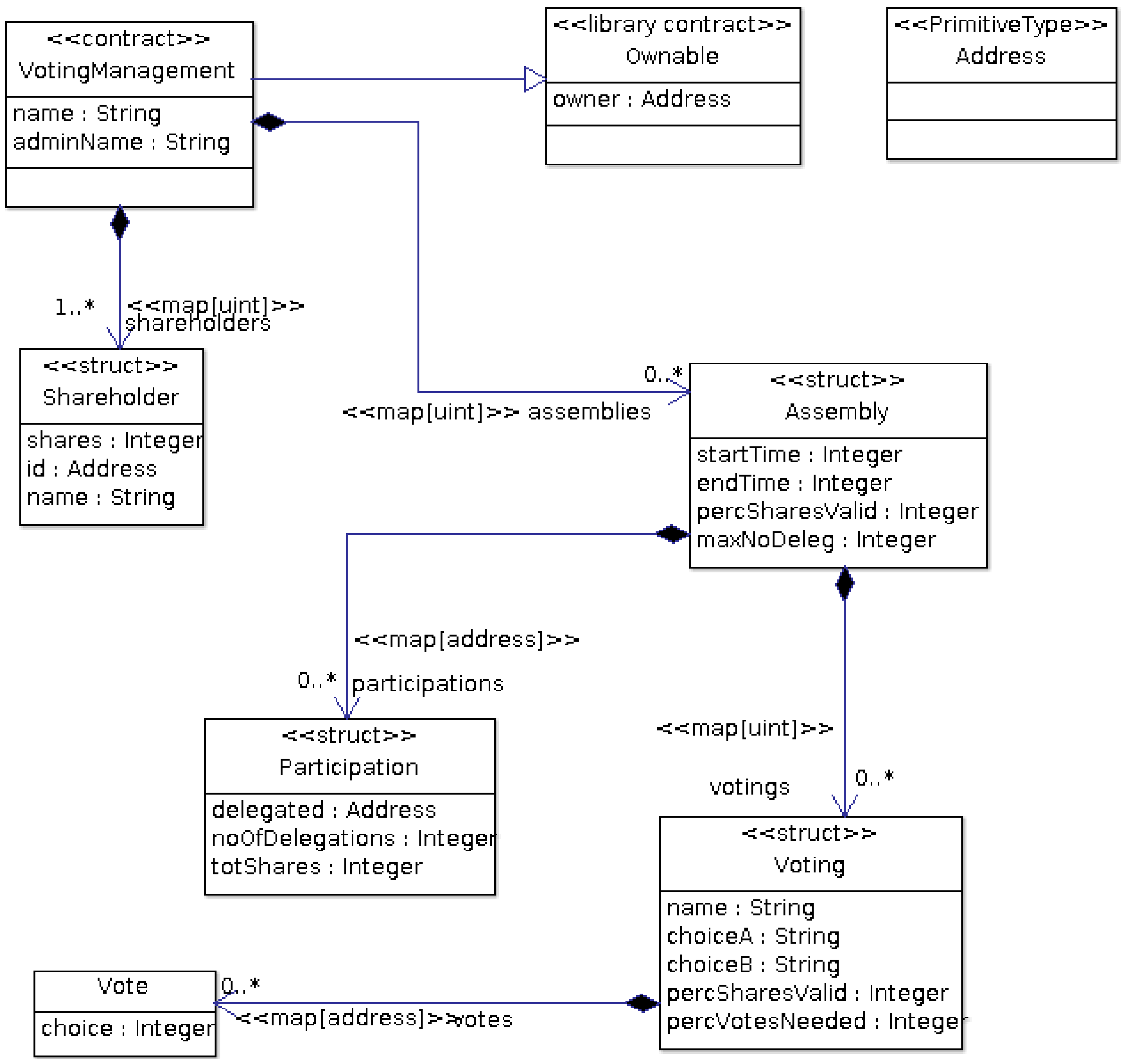
- In this case the subdivision is trivial, because all US make use of the Smart Contracts.
- The DApp subsystem US are the same. Each includes the Blockchain as further Actor.
- The Blockchain subsystem US are the same. The identifiers of the Actors are their unique adresses:
  - Corporate administrator: his address is at first the address that creates the contract, and then possibily a further address set by the Change administrator US.
  - Shareholder: their addresses are specified and managed by the Administrator.

Step 5 – Design of the SC subsystem:

- The system is quite simple, so a single SC looks the best option.
- Following a SC standard, the “`Ownable`” standard abstract contract is used to manage the ownership of the Administrator on the SC:

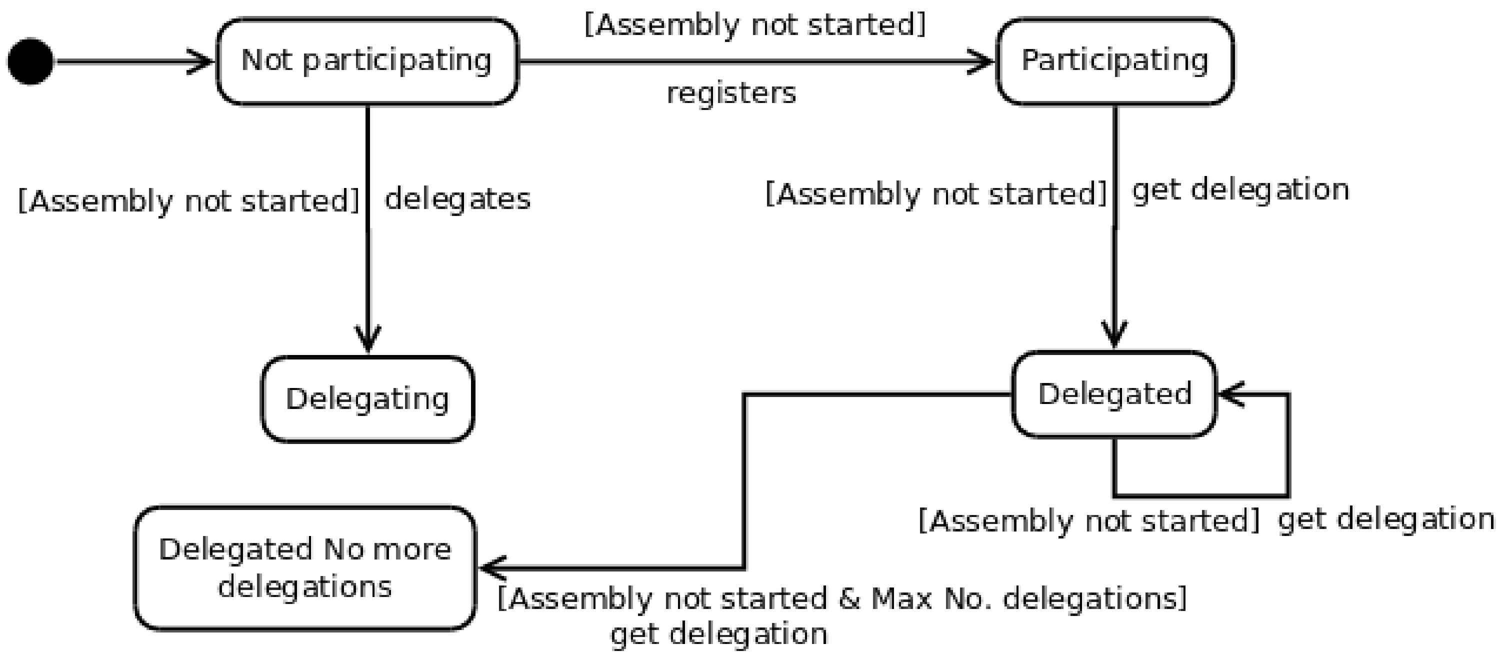
```
contract Ownable {
   address public owner;
   modifier onlyOwner() {
      require(msg.sender == owner);
      _;
   }
}
```

Step 5 – Data structure of the SC shown using a modified UML class diagram:



UML State diagram of a Shareholder

- showing the possible ways of his/her participation to an assembly:



Step 5 – The Dynamic model of the SC subsystem

- **Modifiers:**
  - `onlyOwner()`
  - `onlyShareholder()`
  - `onlyOwnerOrShareholder()`
  - `assemblyRunning()` – enforces that there is actually an assembly running at the time of the call.
  - `assemblyNotRunning()` – enforces that there is no assembly running at the time of the call.

Step 5 – The Dynamic model of the SC subsystem

- **Functions:**
  - `AndSoOnAndSoOn...()`

Conclusions

- Despite the huge effort presently ongoing in developing DApps, software engineering practices are still poorly applied
- A sound software engineering approach might greatly help in overcoming many of the issues plaguing blockchain development:
  - Security issues
  - Software quality and maintenance issues
- Researchers in software engineering have a big opportunity to start studying a field that is very important and brand new
- Blockchain firms, including ICO startups, could develop a competitive advantage using SE practices since the beginning

Thank you for your attention!

Questions?

Michele Marchesi

marchesi@unica.it