Dipartimento di Matematica e Informatica

dApp Implementation using an Agile, yet Systematic Approach

The ABCDE Method: Agile BlockChain Dapp Engineering

Lodovica Marchesi, Michele Marchesi and Roberto Tonelli

Dept. of Mathematics and Computer Science
University of Cagliari, Cagliari, Italy

DMI, Cagliari University, and the Blockchain

- In Cagliari, we started working on Bitcoin and blockchain technologies since February 2014
- Two research groups: Agile Group and the Group on Foundational and applied research on computer security → **blockchain@unica**
- 4 courses for 24 CFU offered on blockchain technology to Computer Science Master students
- Organizer in 2018 and 2019 of four workshops on “Blockchain Oriented Software Engineering”, at PROMISE and ICSE conferences
- More than 10 papers on primary International journals, more than 30 papers on International conferences

DMI, Cagliari University, and the Blockchain

- Organizer of the Scientific School on Blockchain and Distributed Ledger Technologies, Pula, Cagliari on June 2018 and 2019
- Research contracts on blockchain, mainly from Sardinia Region in cooperation with firms, for over 1.5 million Euros
- Possibility given to our CS graduates to certify their Diploma Supplement on the Ethereum blockchain
- A spinoff company, FlossLab ltd working on dApps.
  - They installed an industrial certification system (about 10,000 files/day) on a public blockchain, working since September 2017
  - Probably, the first in Italy

The ABCDE Method

- We have been studying how to introduce sound software engineering practices in dApp and smart contracts development since 2017
- This January, our paper: *Blockchain-oriented software engineering: challenges and new directions, by S.Porru, A.Pinna, M.Marchesi, R.Tonelli,* published in the *Proceedings of the IEEE/ACM 39th International Conference on Software Engineering Companion* has been given the award for being on the most 50 influential papers on blockchain in 2018, at Blockchain Connect Conference: Academic 2019, San Francisco, 11 January 2019
- The first result of these studies is ABCDE!

Agile BlockChain Dapp Engineering

- Based on Agile practices
- Agile practices: proven practices enabling to develop software in the presence of changing requirements
- … but also based on other, more formal approaches:
  - UML diagrams and deep security assessment
- Presently focused on Ethereum Solidity and web3.js development

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

The case for more traditional design

- Key factors in smart contract 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)

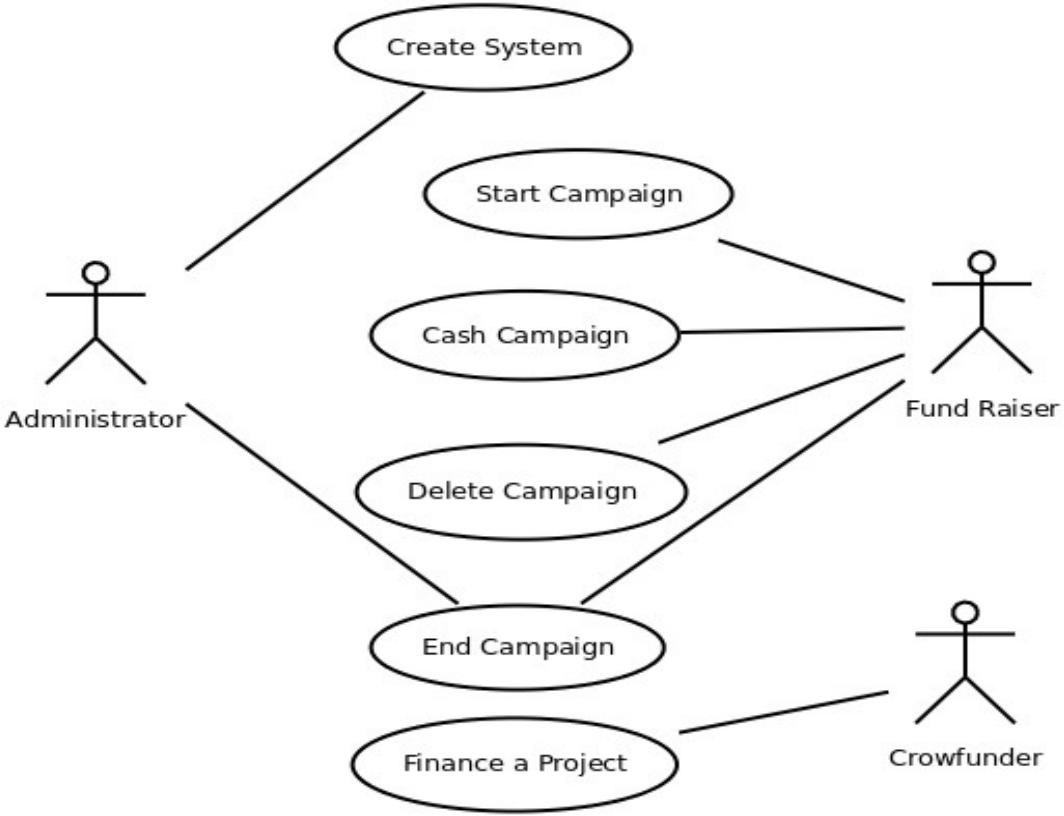
ABCDE Steps

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*

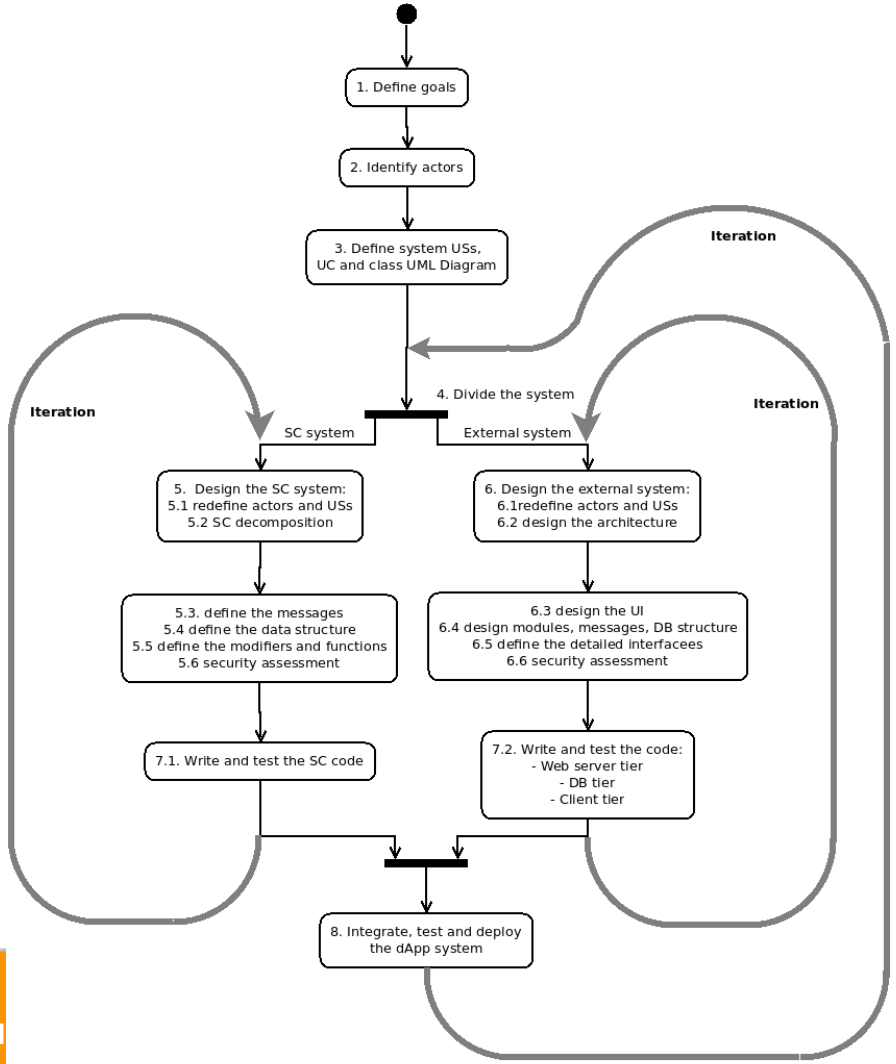
Step 3 – UML Use Case (User Stories) Diagram



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 former, 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

ABCDE Flow


1. Define goals
2. Identify actors
3. Define system USs,
UC and class UML Diagram
Iteration
4. Divide the system
Iteration
Iteration
SC system
External system
5. Design the SC system:
5.1 redefine actors and USs
5.2 SC decomposition
6. Design the external system:
6.1redefine actors and USs
6.2 design the architecture
5.3. define the messages
5.4 define the data structure
5.5 define the modifiers and functions
5.6 security assessment
6.3 design the UI
6.4 design modules, messages, DB structure
6.5 define the detailed interfacees
6.6 security assessment
7.1. Write and test the SC code
7.2. Write and test the code:
- Web server tier
- DB tier
- Client tier
8. Integrate, test and deploy
the dApp system

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)

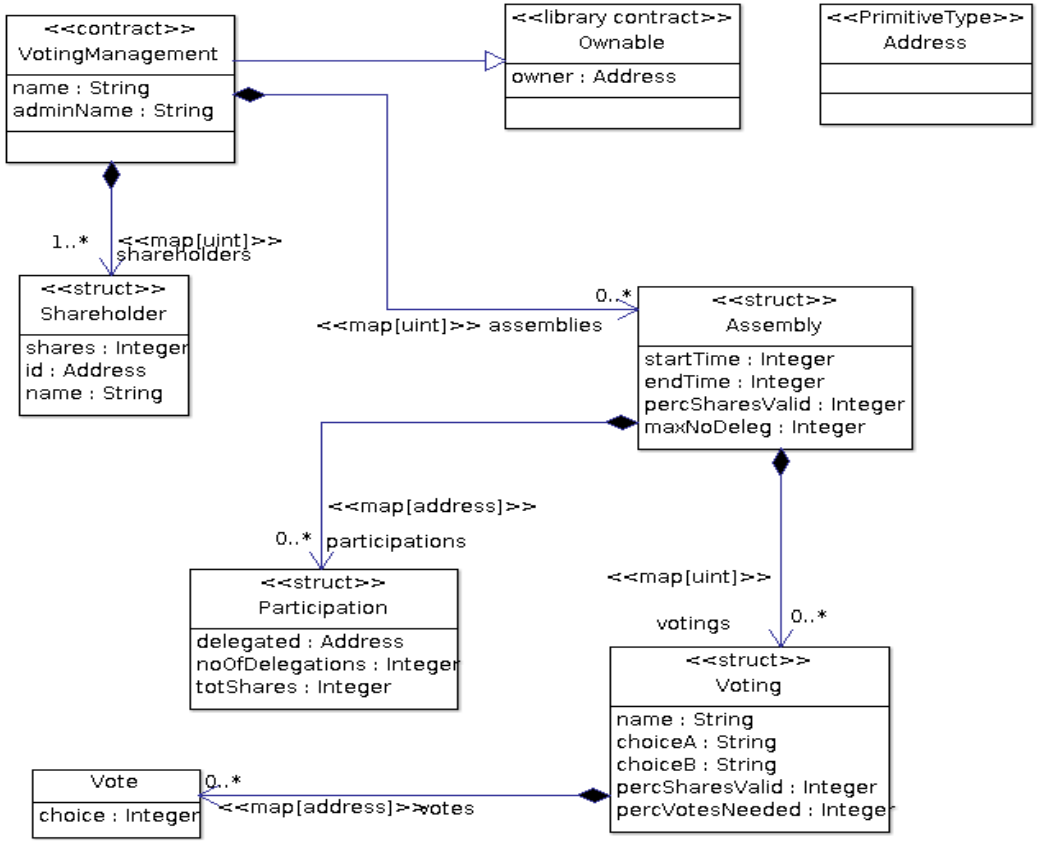
Dynamic SC modeling

- 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

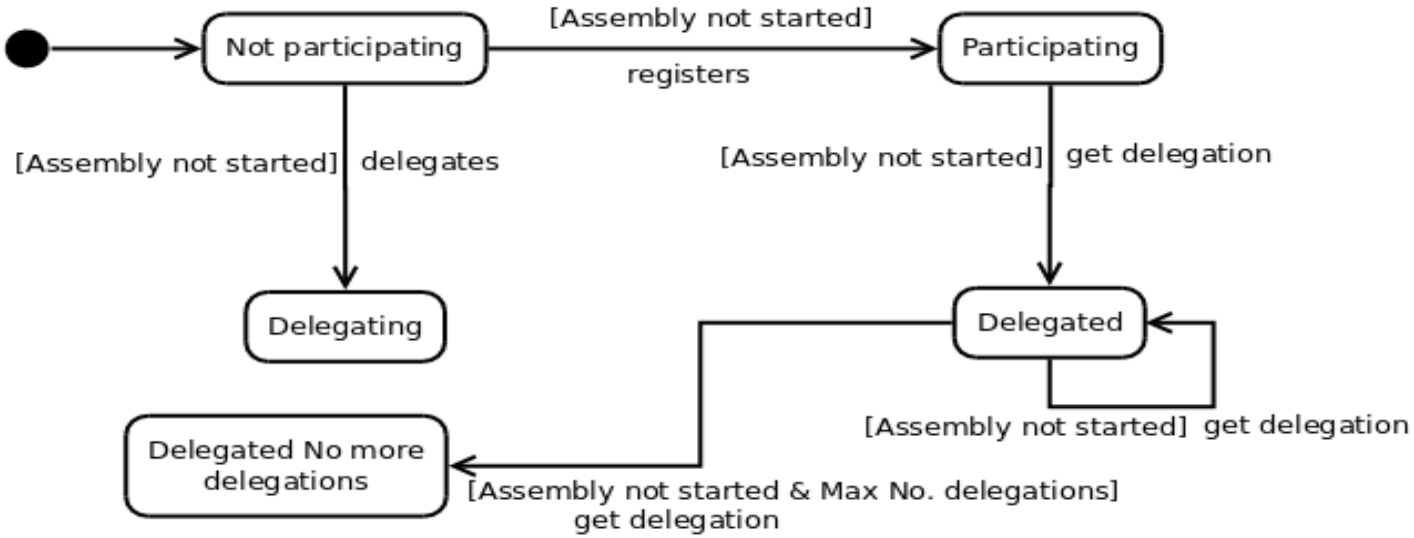
- Apply security patterns/antipatterns
- Use Code reviews
- Use of secure math operations
- Check if Complexity metrics is too high
- Security Analysis services:
  - https://tool.smartdec.net/
  - https://securify.ch/

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

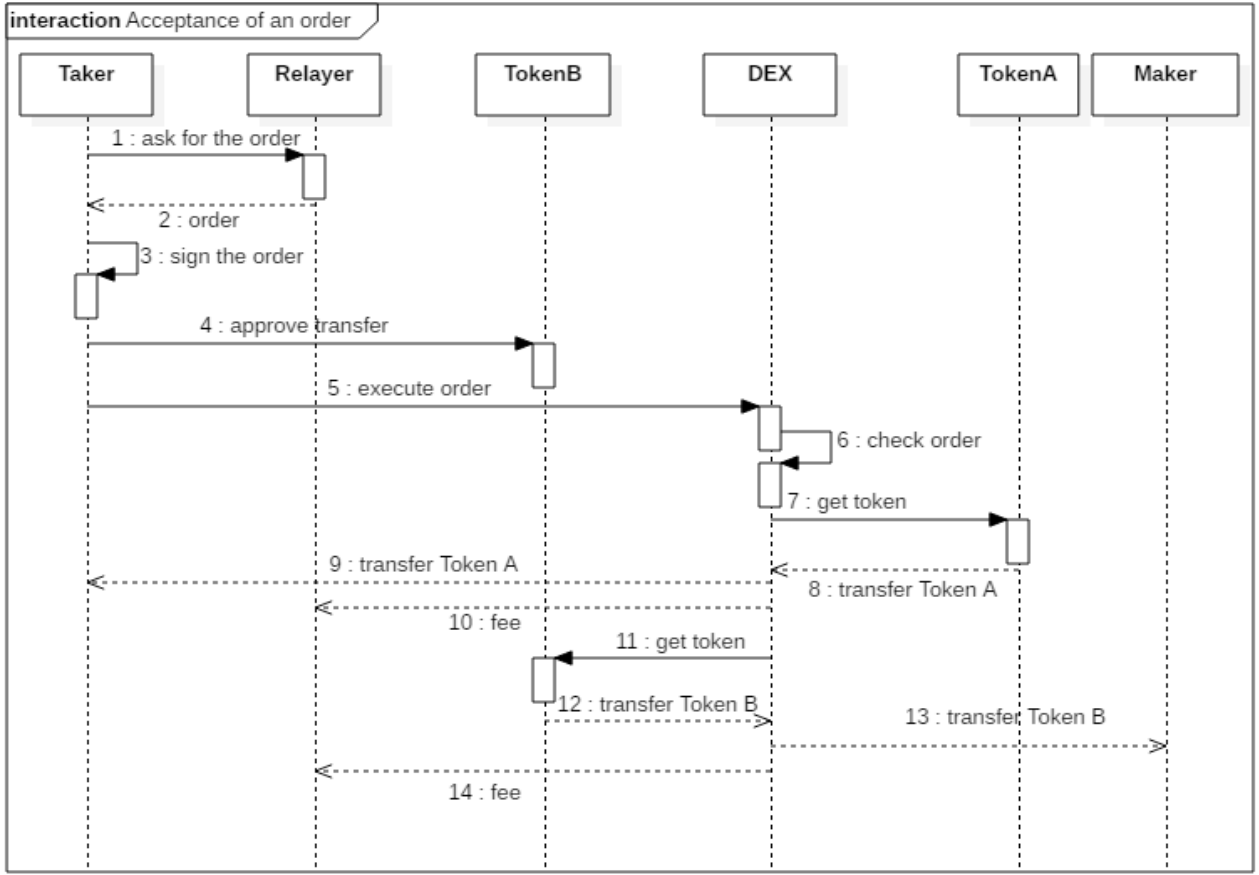


UML State diagram of a Shareholder

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



UML Sequence diagram of EtherDelta DEX



Conclusions

- ABCDE is a first attempt to create a sound software engineering process to specify, design and implement dApps
- It is being used in our group, in our spinoff firm and in other groups we are consulting, with good results
- Future developments:
  - Extend the method beyond Ethereum and Solidity – an extension to Hyperledger Sawtooth is in progress
  - Provide tools to help designing dApps, su as extensions to UML diagrammers