Towards an Approach for Applying Early Testing to Smart Contracts

N. Sánchez-Gómez[a], L. Morales-Trujillo[b] and J. Torres-Valderrama[c]
University of Seville, Escuela Técnica Superior de Ingeniería Informática,
Web Engineering and Early Testing (IWT2) Group, Avda. Reina Mercedes s/n, 41012 Seville, Spain

Keywords: Blockchain (BC), Smart Contracts, Early Testing.

Abstract: Immutability- the ability for a Blockchain (BC) Ledger to remain an unalterable, permanent and indelible history of transactions- is a feature that is highlighted as a key benefit of BC. This ability is very important when several companies work collaboratively to achieve common objectives. This collaboration is usually represented by using business process models. BC is considered as a suitable technology to reduce the complexity of designing these collaborative processes using Smart Contracts. This paper discusses how to combine Model-based Software Development, modelling techniques, such as use cases models and activity diagram models based on Unified Model Languages (UML) in order to simplify and improve the modelling, management and execution of collaborative business processes between multiple companies in the BC network. This paper includes the neccessity of using transformation protocols to obtain Smart Contract code. In addition, it presents systematic mechanisms to evaluate and validate Smart Contract, applying early testing techniques, before deploying the Smart Contract code in the BC network.

1 INTRODUCTION

The first BC that appeared was Bitcoin, when Satoshi Nakamoto (Nakamoto, 2009) released the Version 0.1 of bitcoin software on January 2009. Since then, all BC are based on the same operative. Reviewing the rapid evolution and current state of the BC networks, this technology could have the ability to reconfigure all aspects of today's society. In the logistics industry and Supply Chain (Hackius and Petersen, 2017), BC appears as a facilitator and enabler of operations, because this technology could easily be added to other tools that seek to streamline and optimize the operations of traditional companies.

Smart Contracts (Buterin, 2014) is a related concept to BC Technology (BCT). These are digital contracts that are executed by themselves, without intermediaries, but written as a computer program instead of using a printed document with legal language.

From our point of view, a successful implementation of BCT will only be achieved if it combines the paradigm of Model-based Software Development and modelling techniques in order to simplify and improve the entire business process. In fact, the combination of both techniques has allowed to obtain successful results in different research areas such as requirements engineering (García-García et al., 2012) (Escalona et al., 2013), process management (García-García et al., 2017) or identity reconciliation (Enríquez et al., 2015), among others.

In this context, other important aspect to consider is the Software Testing. Testing has usually been seen as a phase which is always performed at the end once the coding phase is finished and before software is delivered to our customer. But, in the Software Development Life Cycle, software testing should begin as soon as possible, because an early start of the testing phase, helps to reduce the number of defects (Cutilla et al., 2012).

This paper discusses the advantages of applying transformation protocols to obtain Smart Contract code from models. This would also make it possible to apply systematic mechanisms to evaluate and

[a] https://orcid.org/0000-0001-9102-6836
[b] https://orcid.org/0000-0001-9554-1173
[c] https://orcid.org/0000-0002-7786-5841

Sánchez-Gómez, N., Morales-Trujillo, L. and Torres-Valderrama, J.
Towards an Approach for Applying Early Testing to Smart Contracts.
DOI: 10.5220/0008386004450453
In *Proceedings of the 15th International Conference on Web Information Systems and Technologies (WEBIST 2019)*, pages 445-453
ISBN: 978-989-758-386-5

validate the Smart Contract, applying early testing techniques, before deploying the software in the BC network.

The paper is structured as follows: the next section summarizes background and a hypothesis is raised as a starting point (section 2). Then, in Section 3, we present the global approach to solve the identified problem and an overview of our proposed design solution. Finally, Section 4 describes some conclusions and future works.

2 BACKGROUND

2.1 Blockchain Smart Contracts

Nowadays, one of the main trends of IT technology is the so-called BC application. The BC is a technology originally devised to run the Bitcoin cryptocurrency in a decentralized and secure way.

BCT is one form of Distributed Ledger Technology (DLT). A Distributed Ledger is a database that is spread across several computing devices (nodes). Each node replicates and saves an identical copy of the Ledger (so each participant node of the network updates itself independently). The structure of the BCT makes it distinct from other types of distributed ledgers. In this technology, data is grouped together and organized in blocks. In others words, the BC structure is a chronologically ordered list of blocks. These blocks are containers aggregating transactions and, every block, is identifiable and linked to the previous block in the chain (these are then linked to one another secured and immutable using cryptographic techniques).

The Distributed Ledger in a business network is shared and synchronized by the consensus algorithm. Consensus affirms that all parties in this network agrees on the types of information to be captured about an asset (element that is also contained in the BCT). Data provenance is a historical record for any piece of data and this provenance ensures that the parties are able to back trace records of an asset to its origination. On the other hand, the immutability of this technology guarantees that a record in a ledger cannot be removed or altered: when a transaction is committed there is no rolling back, even if it was a mistake.

An immutable and shared BC Ledger is updated every time a transaction occurs through peer to peer replication. The Ledger is distributed and shared so there is no Master control through some centralized mechanism (so each party has a replica ensuring that transactions are secure, authenticated and verifiable).

In this context, the digital contract for asset transference can be embedded in the transaction database. Smart Contracts are the rules that govern a transaction. This is a user-defined program executed on the BC network (Omohundro, 2014). This is the program code that asks the BC to create, delete, modify or return the state of an asset. For a software engineer, a transaction is analogous to a stored procedure call on a database.

An oracle, in the BC context, is an agent that finds and verifies real-world occurrences and submits this information to a BC to be used by Smart Contracts.

BCs can't access data outside their network. An oracle is a data feed – provided by third party service – designed for being used in Smart Contracts on the network BC. Oracles provide external data and trigger Smart Contract executions when pre-defined conditions meet together. These conditions can be validations such as successful payment, package received, etc.

Smart Contracts can be enforced as a part of transactions, and are executed across the BC network by all connected nodes.

The BC platform Ethereum, for example, offers a complete built scripting language for writing Smart Contracts, called Solidity. Its execution environment, the Ethereum Virtual Machine (EVM), comprises all full nodes on the network and executes bytecode compiled from Solidity scripts.

One practical application is the use of BCT for Business Process Management (BPM). It has been discussed under diverse viewpoints. Mendling et al., (2018) about challenges and opportunities and Rosemann and von Brocke (2015) about the challenges and opportunities of BC for BPM in relation to the six BPM core capability areas.

2.2 Early Testing

The goal of software development is to elaborate a product that will bring value to our customers. As a software engineer we must have a reputation for finding and embracing new technologies and programming`s languages. However, these can only take us so far in ensuring the software we build has quality and brings value to our customers.

Therefore, software testing is a very essential part of Software Development Life Cycle (SDLC) and without proper testing software cannot be released to our customers.

Traditionally, software testing has been seen as a phase which is always performed at the end once coding phase is finished and before software is delivered to our customer. But, in the SDLC, software testing should begin as soon as possible. This helps to capture and eliminate defects in the early stages of SDLC. An early start of test, helps to reduce the number of defects and, ultimately, the cost of rework at the end.

This is clearly evidenced, moreover, in the principles of software testing (Graham et al., 2015). "Principle 3: Early testing: To find defects early, testing activities shall be started as early as possible in the software or system development life cycle, and shall be focused on defined objectives. If the testing team is involved right from the beginning of the requirement gathering and analysis phase they have better understanding and insight into the product and moreover the cost of quality will be much less if the defects are found as early as possible rather than later in the development life cycle".

Figure 1, shows an example of the Delayed Issue effect (relating to the relative cost of fixing requirements issues at different phases of a project).

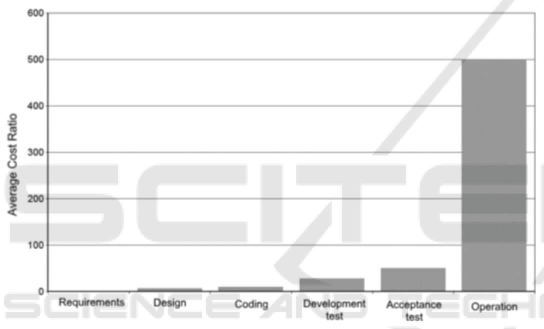

Figure 1: A widely-recreated chart of the Delayed Issue Effect (Boehm, 1981).

It is essential to launch the software testing activity from the collection of objectives and requirements, then refining the software testing in the analysis and design phases. The efficiency of testing as well as the possibility to reduce the overall project time and costs largely depends on how accurately you formulate the requirements to the final software product.

Thus, the final result depends not only on the software engineer but also on our customer. Moreover, the cost to fix an error directly depends on what stage of SDLC has been detected. Any error that is found may cause a domino effect (Rayskiy, 2017). An error that hasn't been found on time may require 100 times more efforts on its fixing after it gets to the stage of software deployment. In this context, the requirements for the final software product are critical (Rayskiy, 2017).

Maximum defects occur in requirement phase. As noted in *"Inspecting Requirements"* (Wiegers, 2001), "Industry data suggests that approximately 50 percent of the product defects are originated in the requirements elicitation. Perhaps 80 percent of the rework effort on a development project can be traced to requirements defects.". This technique would allow the evaluation and validation of the Smart Contract before deploying the software.

In the significant book "Software Testing Techniques" (Beizer, 1990), contains the most complete catalogue of testing techniques, Beizer stated that "the act of designing tests is one of the most effective bug preventers known," which extended the definition of testing to error prevention as well as error detection activities. This led to a classic insight into the power of early testing.

2.3 Model-based Software Development

Since a few years ago, modelling tools help document business processes functionality and through model transformations, partially automate software source code generation using Unified Model Languages (UML) and other modelling standards.

Models, normally, are easier to understand than software source code (Forward and Lethbridge, 2008). Therefore, it also allows improving the productivity of the development and their quality. It's easier to check the correctness of a model and modelling tools can ensure that the deployed code has not been modified after its generation from the model (Lu et al., 2018).

In comparison to traditional software development (Figure 2), where phases are clearly separate, Model- based Software Development shows the phases specification, design and implementation to have grown together much more strongly (Conrad et al., 2005).

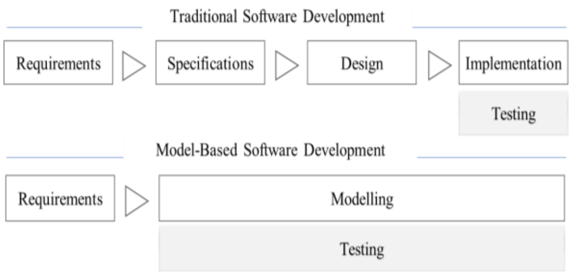

Figure 2: Traditional vs Model-Based Software Development (Conrad et al., 2005).

Model-based Design combined traditional Software Development and Systems Engineering best practices with visual modelling best practices. Model-based Design principles and best practices continue to evolve, but they are complete architecture blueprint, organized as a framework with multiple viewpoints as the primary work artefact throughout the SDLC.

The major advantages of a Model-Based Design approach by technology driver are (PivotPoint Technology™, 2019):

- Requirements are an integral part of model and other parts of the model can be traced back to requirement.
- Provide a precise architecture blueprint organized by views/viewpoints that are meaningful to all systems stakeholders.
- Automate system validation and verification (reduce errors in the life cycle), automate generation of quality code and automate testing (ensure system implementation is correct and reliable).

Model-based Testing is a testing technique where the runtime behaviour of an implementation under test is checked against predictions made by a formal specification, or model (Pretschner et al., 2005).

In the context of Blockchain-oriented applications, Model-based Software Development is of particular importance for the following reasons (Lu et al., 2018):

- Model-Based tools can implement best practices and generate well-tested code, thereby reducing the occurrence of vulnerable code.
- Models can avoid lock-in to specific BCT since they can be platform-agnostic, and model-based (Model-based tools can be applied at multiple BC platforms).
- Models are easier to understand than code. It is easier to check the correctness of a model and model-based tools can ensure that the deployed code has not been modified after its generation from the model.

2.4 Our Hypothesis

BC is an immutable, transparent and secure technology (Sultan K., 2018) for recording the state and ownership of an asset. These assets can be something tangible and diverse as Internet of Things (IoT) devices, a piece of art, ... or they can also be something intangible (e.g. intellectual property).

This Technology allows unknown or untrustworthy parties to conduct transactions efficiently and accurately. It automates the contractual agreements between stakeholders in a business network by the use of Smart Contract.

As indicated in subsection 2.1, Smart Contracts are digital contracts that are executed by themselves, without intermediaries, but these are written as a computer program instead of using a printed document with legal language. That is, a Smart Contract is a set of formal rules under which the parties to that Smart Contract agree to interact with each other and complete a transaction.

Specifically, a Smart Contract is a computer program which is executed after the completion of a transaction and, through this, the logical rules (Boolean function as if-then-else) are defined in the same way as a traditional legal contract would, indicating the agreements and obligations (and possible sanctions) that can occur in various circumstances.

Smart Contracts for Ethereum are typically written using the Solidity language. Solidity is an object-oriented language, and the contracts are defined in it like classes – they have a data structure, public and private functions, and can inherit from other contracts. Smart Contracts have also specific concepts like events and modifiers.

Currently, Smart Contracts can be programmed in others numerous languages, such as JavaScript, Go, Python, C #, Ruby, PHP, Scala, etc.

There are even initiatives that allows exploring and interacting with digital contracts using a REST API (Application Programming Interface). For example, ETHEREST Ethereum Contract API is a tool that provides a way to interact with Smart Contracts using the user interface of the website or the API. APIs are already used on all BCs because of their advantage to make function coding's much easier.

Given the immutability of BCT, It is essential that, before deploying a Smart Contract code in a business network, they go through evaluation and validation processes. A defect of Smart Contract may cause a non-repairable effect.

In order to achieve this objective, it is necessary that user requirements, use cases, activity diagrams, etc. are an integral part of model and to provide a precise architecture blueprint, organized by views/viewpoints, that are meaningful to all systems stakeholders.

Since Smart Contracts have some very specific characteristics, it is necessary to introduce some new concepts in these diagrams, to be able to better model and specify Smart Contracts. Whenever possible, these concepts are simply introduced as UML stereotypes, which are tags that can be used in UML diagrams wherever needed. In a few other cases, it is sufficient to introduce a specific notation like the transfer of cryptocoins in sequence or activity diagram (Marchesi et al., 2018).

Nowadays, the platforms BC Smart Contracts as Ethereum, NEM, NEO, etc., allow anyone to build and execute Smart Contract code directly, without following good practices of software engineering without going through evaluation and validation processes.

This means software developers may run Smart Contracts with bugs and serious security vulnerabilities (to understand the severity of the problem, can see this list of known bugs and vulnerabilities from https://consensys.net (Ethereum Smart Contract Best Practices).

A handful of companies, e.g. Solidified™ (https://solidified.io/) is one of the largest auditing platform for Smart Contracts have stepped forth to provide auditing services for Smart Contracts) reviewing the code and providing feedback on its quality and security, but it is crucial to start the software testing activity before having software's code source, it should be from the requirements and then refine the software testing in the phases of analysis and design (throughout the typical SDLC).

The SDLC is a framework that defines the tasks to be performed at each step of the software development process.

The life cycle of Smart Contract development must also clearly define the methodology for writing and improving the quality of this software and the overall development process (The modex Team, 2019), by streamlining the use cases writing process, the activity diagrams writing process and the code writing process as well as the early testing process.

In this context, this hypothesis is raised: Smart Contracts model-based can improve the verification and validation of Smart Contracts code through the application of testing techniques from the early stages of the SDLC, what is known as early testing.

3 TOWARDS A MODEL-BASED BLOCKCHAIN ENGINEERING

As mentioned above, following a model-based approach helps to derive an understanding of a system, by bringing together different views with various levels of abstraction.

According to Seebache and Maleshkova (2018), Model-Based Engineering contributes to describe and understand a system in various ways:

- Since a model builds upon a well-defined notation and typology, the relationships between the distinct elements as well as their descriptions contribute to a general understanding of the system, while helping to develop scalable solutions as well.
- An architectural framework may be used to combine and transform different models and descriptive layers to facilitate the construction of a system.
- Building upon a set of formalized meta-models, which in turn can be integrated and transformed into models with a higher degree of information, automation may be applied.

As shown in Figure 3, Model-Based Engineering can be applied both on System modelling and Test modelling.

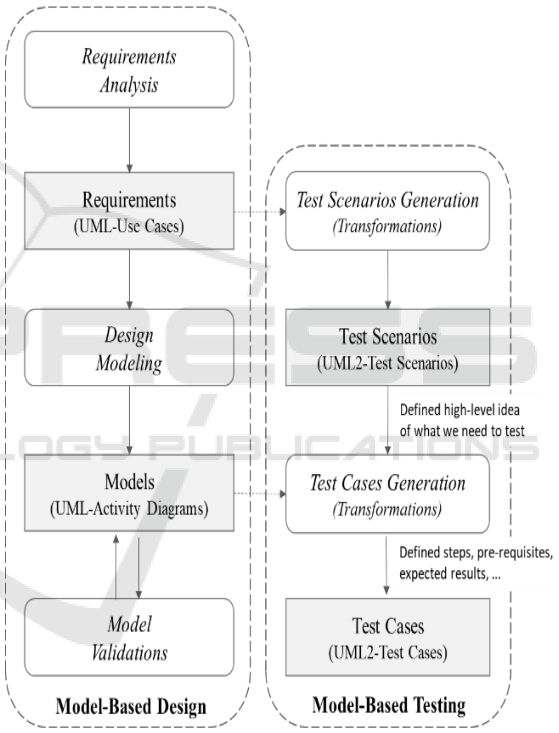
Requirements Analysis
Requirements (UML-Use Cases)
Design Modeling
Models (UML-Activity Diagrams)
Model Validations
Model-Based Design
Test Scenarios Generation (Transformations)
Test Scenarios (UML2-Test Scenarios)
Defined high-level idea of what we need to test
Test Cases Generation (Transformations)
Defined steps, pre-requisites, expected results, ...
Test Cases (UML2-Test Cases)
Model-Based Testing

Figure 3: Model-Based Engineering.

User requirements are represented in a tabular format, which may facilitate requirements tracing during the system life cycle. This is important to know what happens when related requirements change or are deleted, which improves traceability.

Nowadays, requirements and use cases are a widely used technique to define the functional requirements of software systems. Several authors, like Escalona et al., (2006), García-García et al., (2012), Achour (1998) or Cockburn (2000) propose how to define use cases with UML.

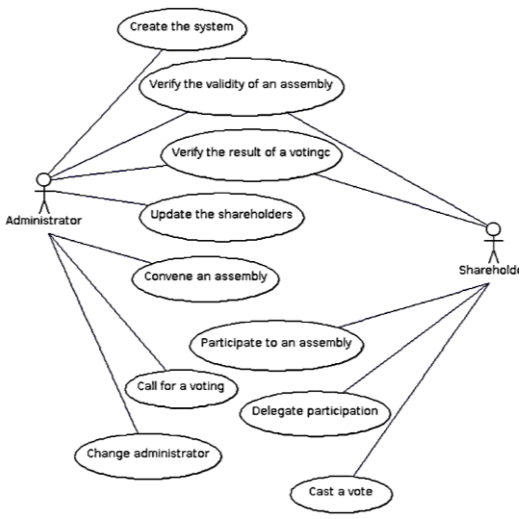

Figure 4: Use case diagram template (Marchesi et al., 2018).

Figure 4, shows an example use case diagrams, which describes the relations between use cases and actor (these diagrams can also describes the relation between use cases and other use cases). Specifically, this diagrams describing the behaviour of every use case and their pre-conditions, post-conditions, priority, etc.

The UML is a visual language to support the design and development of complex systems. But UML itself, even the newest version 2.0 (Binder, 2000), provides no means to describe a test model.

Specifically, UML 2.0 for testing, called UML 2.0 Testing Profile (U2TP) (Cockburn A., 2000) would close the gap between designers and testers by providing a meaning to use UML for both system modelling and test specification. This allows a reuse of UML design documents for testing and enables test development in an early system development phase.

Model-Based Design can implement best practices and generate test cases from the early stages of the software development lifecycle, thereby reducing the number of bugs when coding.

Test Scenarios are derived from the UML-Use case and Test Cases are derived from the UML-Activity diagrams.

Test Cases are obtained through transformations, implements the all nodes, all transitions, and all criteria to select scenarios. They select the paths that go across a higher number of actions until all the actions of the activity diagrams have been traversed at least once. For the all-transitions criterion, they select the path that traverse a higher number of object-flow edges until all of them have been crossed at least once.

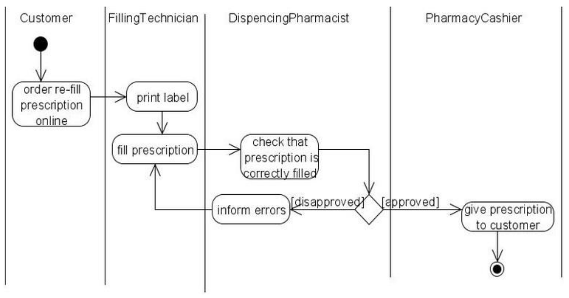

Figure 5: Business process template in UML-Activity diagram (Donyina and Heckel, 2009).

Figure 5, shows a UML-Activity diagram. If the activity diagram has not got any loops, the all-scenarios criterion selects the paths that go through all output object-flow edges from decision nodes at least once. If the activity diagram has got some loops, the all-scenarios criterion selects the paths that go through all output object-flow edges from decision nodes and all combinations among loops at least once.

And ultimately, it could generate Smart Contracts code and REST API following the REST principles (Fielding and Taylor, 2002) with different roles in BC as a System of Connectivity, as a System of Security, as a System of Chain Management, etc. (Sandoval, 2018).

Therefore, we can adopt Model-based Software Development to facilitate the development of BC applications in the space of business processes.

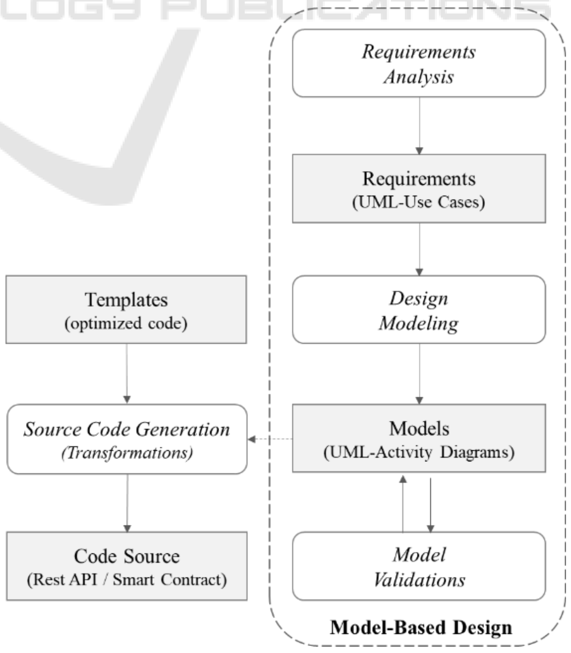
Requirements Analysis
Requirements (UML-Use Cases)
Design Modeling
Models (UML-Activity Diagrams)
Model Validations
Model-Based Design
Templates (optimized code)
Source Code Generation (Transformations)
Code Source (Rest API / Smart Contract)

Figure 6: Source Code Generation.

Figure 6 illustrates our proposal to generate source code, which consists of the following basic elements: Smart Contracts templates, UML-Activity diagrams, a Smart Contracts generator and a BC triggers through the use of REST API.

Smart Contract templates are based on the framework of Grigg's Ricardian Contract triple of "prose, parameters and code" (Grigg, 2004) (Grigg, 2015). The first application to implement Ricardian contracts is OpenBazaar (https://openbazaar.org/), a peer-to-peer e-commerce platform where you can trade almost anything directly with each other.

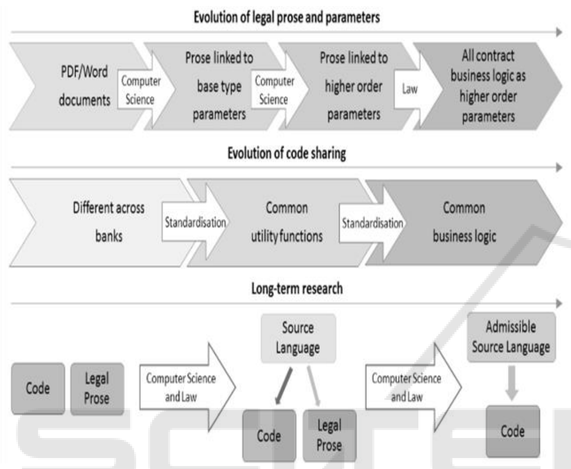

Figure 7: Potential evolution of important aspects of legally-enforceable Smart Contracts (Clack et al., 2016).

Figure 7 illustrates the potential evolution of important aspects of legally – enforceable Smart Contracts: legal prose and parameters, code sharing, and long-term research. It can easily be adapted to specific use cases and, in addition, could support legally-enforceable Smart Contracts using operational parameters to connect legal agreements to standardised code (Clack et al., 2016).

There are already initiatives, such as Lorikeet (Tran et al., 2018), where the tool developed can automatically create Smart Contract code from specifications that are encoded in the business process and data registry models based on the implemented model transformations.

With these pillars, we can automatically create Smart Contracts in, for example, Solidity from UML-Activity diagram models (because UML-Use case and Test Cases are derived from the UML-Activity diagrams) and REST API to provide data and trigger Smart Contract executions. The REST API is necessary to expose the BC logic to web or mobile applications, as well as integrating the BC with existing enterprise systems of record or legacy system.

Consequently, this automation will reduce the level of inherent complexity associated with BC Smart Contracts and encourage their adoption across a diverse domain (banking, finance, real estate, governance, education, entertainment, …) of applications.

4 CONCLUSIONS AND FUTURE WORKS

This paper has presented a discussion about the technological constraints and current situation of the BC Smart Contract. As it introduces, BCT is a fascinating technology that is producing a revolution in ICT (Information and Communication Technologies), but the necessity of assuring software quality of a BCT is a current unsolved problem.

In this paper, we present the problem in detail and introduce a preliminary view of an approach based on the use of models and the necessity of having a complete framework for orchestrating the life cycle of BC Smart Contract.

Tools or techniques for modelling and managing the peculiarities a software engineer must face when dealing with BC oriented software systems are still matter for researchers. Tools and techniques of traditional software engineering have not yet been adapted and modified to adhere to this new software paradigm.

A sound software engineering approach might greatly help in overcoming many of the issues plaguing BC development providing software engineer with instruments similar to those typic used in traditional software engineering to afford architectural design, security issues, testing planes and strategies to improve software quality and maintenance (Marchesi et al., 2018).

It is clear that we have many future objectives. The first one is to study the current situation of source code generator tools and early testing in network BC. To this end, a Systematic Literature Review (SLR) is being developed following the approach of Kitchenham and Brereton (2013). We want to focus our work in the field of the Model-Driven paradigm but obviously, it depends on our previous results.

Another important future work is trying to make a proposal based on the previous idea (Figure 3 and 6) and test it in the industry. Currently, our research group has numerous contacts with different companies that are already working on this topic and we can propose and address projects that allow us to test and validate our work.

ACKNOWLEDGEMENTS

First of all, we would like to thank all experts for their participation and sharing their valuable knowledge. Moreover, we would like to thank all participants in our pretests for their collaboration.

This research has been supported by POLOLAS Project (TIN2016-76956-C3-2-R) of the Spanish the Ministry of Economy and Competitiveness.

REFERENCES

Achour C.B., 1998. Writing and Correcting Textual Scenarios for System Design. *Natural Language and Information Systems Workshop*. Vienna, Austria.

Beizer B., 1990. Software Testing Techniques. *Van Nostrand Reinhold Company Limited*.

Binder R. V., 2000. Testing Object-Oriented Systems. *Addison-Wesley*. USA.

Boehm B., 1981. Software Engineering Economics. *Prentice Hall*, Englewood Cliffs, NJ.

Buterin V., 2014. Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Pape.

Clack C.D., Bakshi V.A., Braine, L., 2016. *Smart Contract Templates: essential requirements and design options.* © Barclays Bank PLC 2016. This work is licensed under a Creative Commons Attribution 4.0 International License.

Cockburn, A. 2000. *Writing Effective Use Cases*. Addison-Wesley 1st edition. USA.

Conrad, M., Fey, I., Sadeghipour, S. 2005. Systematic Model-Based Testing of Embedded Automotive Software. *Electronic Notes in Theoretical Computer Science (Book)*.

Cutilla, C. R., García-García, J. A., Gutiérrez, J. J., Domínguez-Mayo, P., Cuaresma, M. J. E., Rodríguez-Catalán, L., & Mayo, F. J. D., 2012. Model-driven Test Engineering-A Practical Analysis in the AQUA-WS Project. In *ICSOFT* (pp. 111-119).

Donyina A., Heckel R., 2009. Formal Visual Modelling of Human Agents in Service Oriented Systems. 2009 *Fourth South-East European Workshop on Formal Methods*.

Enríquez, J. G., Domínguez-Mayo, F. J., Escalona, M. J., García García, J. A., Lee, V., & Goto, M., 2015. Entity Identity Reconciliation based Big Data Federation-A MDE approach.

Escalona M.J., Gutiérrez J.J., Villadiego. D., León. A., Torres A.H., 2006. Practical Experiences in Web Engineering. *15th International Conference On Information Systems Development*. Budapest (Hungary).

Escalona, M. J., Urbieta, M., Rossi, G., Garcia-Garcia, J. A., & Luna, E. R., 2013. Detecting Web requirements conflicts and inconsistencies under a model-based perspective. *Journal of Systems and Software*, 86(12), 3024-3038.

Fielding, R. T., Taylor R.N., 2002. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, Vol. 2,

Forward, A., Lethbridge, T., 2008. Problems and opportunities for model-centric versus code-centric software development: A survey of software professionals. *International Workshop on Models in Software Engineering*.

García-García, J. A., Escalona, M. J., Ravel, E., Rossi, G., & Urbieta, M., 2012. NDT-merge: a future tool for conciliating software requirements in MDE environments. In *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services* (pp. 177-186). ACM.

García-García, J. A., Enríquez, J. G., García-Borgoñón, L., Arévalo, C., & Morillo, E., 2017. A MDE-based framework to improve the process management: the EMPOWER project. *In 2017 IEEE 15th International Conference on Industrial Informatics (INDIN)* (pp. 553-558). IEEE.

García-García, J.A., Ortega, M.A., García-Borgoñón, L., Escalona, M.J., 2012. NDT-Suite: a model-based suite for the application of NDT. In *International Conference on Web Engineering*. Springer, Berlin, Heidelberg.

Graham D., Van Veenendaal E., Evans I., Black R., 2015. Foundations of Software Testing: ISTQB Certification Cengage Learning Emea; Revised edition.

Grigg, I. 2004. The Ricardian Contract. In *Proceedings of the First IEEE International Workshop on Electronic Contracting.* http://iang.org/papers/ricardian_contract.html.

Grigg, I. 2015. The Sum of All Chains — Let's Converge!, *Presentation for Coinscrum and Proof of Work*. http://financialcryptography.com/mt/archives/001556.html.

Hackius, N.; Petersen, M., 2017. Blockchain in Logistics and Supply Chain: Trick or Treat?. In *Proceedings of the Hamburg International Conference of Logistics (HICL)*, Hamburg, Germany.

Kitchenham B., Brereton P., 2013. *A systematic review of systematic review process research in software engineering.* Information *& Software* Technology.

Lu Q, Weber I, Staples M., 2018. *Why Model-Driven Engineering Fits the Needs for Blockchain Application Development.* IEEE Blockchain Technical Briefs, September 2018.

Marchesi M., Marchesi L., Tonelli R., 2018. *An Agile Software Engineering Method to Design Blockchain Applications.* Software Engineering Conference Russia (SECR 2018). Moscow (Russia).

Mendling, J., Weber, I., van der Aalst, W.M.P., vom Brocke, J., Cabanillas, C., Daniel, F., Debois, S., Di

Ciccio, C., Dumas, M., Dustdar, S., Gal, A., Garcıa-Banuelos, L., Governatori, G., Hull, R., Rosa, M.L., Leopold, H., Leymann, F., Recker, J., Reichert, M., Reijers, H.A., Rinderle-Ma, S., Solti, A., Rosemann, M., Schulte, S., Singh, M.P., Slaats, T., Staples, M., Weber, B., Weidlich, M., Weske, M., Xu, X., Zhu, L. 2018. Blockchains for business process management - challenges and opportunities. *ACM Transactions on Management Information Systems (TMIS)*. Volume 9 Issue 1.

Nakamoto, S. 2009: *Bitcoin: A Peer-to-Peer Electronic Cash System.*

Omohundro, S, 2014. Cryptocurrencies, smart contracts, and artificial intelligence. Published in *AI Matters*.

PivotPoint Technology™, 2019. Digital Engineering Solutions^{SM} for Wicked Problems (https://Pivotpt.com)

Pretschner A., Prenninger W., Wagner S., Kuhnel C., Baumgartner M., Sostawa B., Z¨olch R., Stauner T., 2005. One evaluation of model based testing and its automation. *ICSE'05.*

Rayskiy A., 2017. *Why Should Testing Start Early in Software Project Development?.* https://xbsoftware.com/blog/why-should-testing-start-early-software-project-development/

Rosemann, M., von Brocke, J., 2015: The Six Core Elements of Business Process Management. *Handbook on Business Process Management 1*. Springer.

Sandoval K, 2018. The Role of APIs In Blockchain. https://nordicapis.com/the-role-of-apis-in-blockchain/. Bloc Nordic APIs.

Seebache S., Maleshkova M., 2018. Model-driven Approach for the Description of Blockchain Business Networks. *Proceedings of the 51st Hawaii International Conference on System Sciences*.

Sultan K., Ruhi U., Lakhani R., 2018. Conceptualizing Blockchains: Characteristics & Applications. *11th IADIS International Conference Information Systems*.

The modex Team, 2019. The life cycle of Smart Contract development https://blog.modex.tech/the-life-cycle-of-smart-contract-development-58b04f65de09

Tran AB, Lu Q, Weber I., 2018. Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management. In: *BPM Demos*. CEUR-WS.

Wiegers K.E., 2001. Inspecting Requirements. *StickyMinds.com Weekly Column*.