The Importance of Testing in the Early Stages of Smart Contract Development Life Cycle

N. Sánchez-Gómez*, L. Morales-Trujillo, J. J. Gutiérrez
and J. Torres-Valderrama

University of Seville, Escuela Técnica Superior de Ingeniería Informática. Web Engineering and Early Testing (IWT2) Group. Avenida Reina Mercedes s/n. 41012 Sevilla, Spain
E-mail: nicolas.sanchez@iwt2.org; leticia.morales@iwt2.org; javierj@us.es; jtorres@us.es
**Corresponding Author*

Received 20 December 2019; Accepted 14 April 2020;
Publication 03 June 2020

Abstract

The use of smart contract augurs a world without intermediaries because the code and the agreements contained therein exist across a distributed, decentralized blockchain network. In software engineering, this collaboration is usually represented by using business process models and smart contracts can be used to implement business collaborations in general and inter-organizational business processes. The validation of this contract and the assurance of its quality are critical for its right application. Early testing in smart contract definition is the fact of this paper. The paper discusses the possibility to use transformation protocols to obtain derived artefacts like test case definitions and smart contract code scaffolds. Generation of derived artefacts significantly reduces the number of defects before deploying the smart contract code in the blockchain network. Transformations protocols are created using model-based software development and modelling techniques. This approach allows to simplify and improve the management and execution

Journal of Web Engineering, Vol. 19_2, 215–242.
doi: 10.13052/jwe1540-9589.1925

of collaborative business processes. This would allow, in addition, the application of systematic mechanisms to evaluate and validate the smart contract and, particularly, the application of early testing techniques which would help to reduce the number of defects and, ultimately, the cost of the final review.

Keywords: Blockchain, Smart contract, Model-based software development, Early testing.

1 Introduction

Blockchain is a concept that was first used int the context of Bitcoin. Satoshi Nakamoto [32] released the version 0.1 of bitcoin software on January 2009. Currently, blockchain is being applied in a lot of different contexts using the same operative.

Reviewing the rapid evolution and current state of the blockchain networks, this technology can reconfigure all aspects of today society. For example, in the logistic industry and Supply Chain [27], blockchain appears as a facilitator and enabler of operations, because it could easily be added to other tools that seek to streamline and optimize the operations of traditional companies. This technology offers us a challenge to improve how we used them and how we guarantee the right application of their principles. Blockchains present some specific concepts and concrete architectures that require special contributions, also in the context of software engineering.

One of the key factors of blockchain is smart contract [5]. A smart contract is a digital intelligent virtual contract that defines the constraints, preconditions and postcondition that a blockchain follows. It is a digital contract that are executed by themselves, without intermediaries, but written as a computer program instead of using a printed document with legal language and it assures the blockchain user what this blockchain offers.

With this introduction, we can deduce that smart contracts are critical elements in the context of blockchain. It is important to dedicate efforts and resources to guarantee their quality and coherence. However, the definition and mainly the validation of smart contract is still a fact to improve in the context of software engineering.

Other important aspect to consider, in this context, is the Software Testing. Testing has usually been 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 (SDLC), software testing should begin as soon as possible, because an early start of the testing phase helps to reduce the number of defects [10].

From our point of view, one of the most successful implementations of blockchain technology could combine the paradigm of model-based software development and modelling techniques to simplify and improve the entire business process, mainly the early testing. In fact, the combination of both techniques has allowed to obtain successful results in different research areas such as requirements engineering [15, 19], process management [20] or identity reconciliation [13], among others.

Main contribution of this paper is a discussion about the advantages of applying transformation protocols using model-based development to obtain smart contract code from smart contract definition as a model. This allows to apply systematic mechanisms to evaluate and validate the smart contract, applying early testing techniques, before deploying the software in the blockchain network. When any new defect is found in final stage of testing, it may be necessary to update the design and analysis of the software. Thus, it is important to perform testing in every phase because it gives us confidence that software will run according to the expectation and will not fail once it gets delivered to the client. Early testing will give enough time to recognize absent and inadequate functional requirements.

The paper is organized as follows: the next section summarizes the context (Section 2) where the concepts in which our approach is based are presented. This continues with Section 3 where the hypothesis is raised as a starting point. Then, in Section 4, we present the global approach to solve the identified problem and an overview of our proposed design solution. Finally, Section 5 describes some conclusions and future work.

2 Context

In this Section, different concepts and technologies that are considered in our approach are presented. Firstly, blockchain technologies are presented and the smart contract concept is analyses. The Section continues with a vision of blockchain oracles and finalises with software engineering, software testing and model-driven engineering principles presentation. As it is presented in Section 3, these concepts and technologies are the base of our proposal.

2.1 Blockchain and smart contract

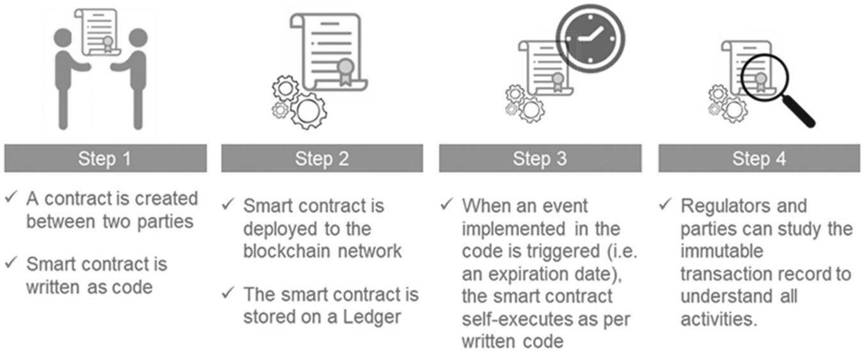
Nowadays, one of the main ICT trends is the so-called blockchain application. Blockchain is a technology originally conceived to run the Bitcoin cryptocurrency in a decentralized and secure way. This technology is one form of Distributed Ledger Technology (DLT). A Distributed Ledger is a database that is spread across several computing devices (nodes). Each node replicates and stores an identical copy of the Ledger.

In blockchain technology, data is grouped and organized in blocks. These blocks are transactions aggregation containers. Every block is identifiable and linked to the previous block in the chain. These blocks are securely and immutably linked using cryptographic techniques. 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.

One of the basic concepts that rules the blockchain technology is the concept of smart contracts [51]. Smart contracts represent the idea of defining constraints, preconditions and postconditions that a concrete block in a blockchain must guarantee that it follows. It gives us new ways to formalize the digital relationships which are much more functional than their paper-based ancestors. In fact, a smart contract is a digital contract to define these aspects. For instance, if a blockchain is the database, then the smart contracts are the rules that govern a transaction. This is a user-defined program executed on the blockchain network [35]. This is the program code that asks the blockchain network to create, remove, modify, or return the state of an asset.

Figure 1 shows a self-explanatory illustration of the use of smart contracts:

Therefore, using blockchain technology, untrusted parties can establish trust in the truthful execution of the code. Smart contracts can be used to implement business collaborations in general and inter-organizational business processes in particular. The potential of blockchain-based distributed ledgers, to enable collaboration in open environments, has been successfully



Step 1

- ✓ A contract is created between two parties
- ✓ Smart contract is written as code

Step 2

- ✓ Smart contract is deployed to the blockchain network
- ✓ The smart contract is stored on a Ledger

Step 3

- ✓ When an event implemented in the code is triggered (i.e. an expiration date), the smart contract self-executes as per written code

Step 4

- ✓ Regulators and parties can study the immutable transaction record to understand all activities.

Figure 1 How a smart contract works?

tested in diverse fields [49]. Several blockchains are currently adopted in various domains to facilitate the operation of new business processes [33, 44].

Smart contracts for Ethereum are typically written using the Solidity language.[1] 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 be inherited from other programs. They also have specific concepts such as events and modifiers.

The main use of smart contracts is tokenization. The process of issuing ownership rights to real-world assets or utilities in the form of a token. One of the main reasons for the growth of tokenization is that it does not require off-chain data. All token information is already known and stored in the blockchain.

In contrast, smart contracts for industries or public sector need external off-chain data such as IoT data, Citizenship data, and events data to trigger execution. This trigger data is not stored on the same blockchain as the smart contract, because it is simply neither realistic to do so.

2.2 Blockchain oracle

Since a smart contract often needs access to information from the outside world, which is relevant to the contractual agreement, so-called oracles were created. These oracles are services that send and verify real-world occurrences and submit this information to smart contracts, triggering changes of state in the blockchain.

Therefore, an oracle is a blockchain middleware that creates a secure connection between smart contracts and various off-chain resources that they need to function. It acts as the middle layer between a blockchain and an API (application programming interface) that translates information for the blockchain to read.

In order to illustrate this idea, Figure 2 shows an API server composed of different REST API (the most-used web service technology) such as SMS (Short Message Service) API, GPS (Global Positioning System) API, etc.

This is the workflow of the ideal oracle, that is, the smart contract sends the request to the Oracle contract, and obtains the external data through the API interface, more specifically, the external data is given to the Oracle in the chain, and then Oracle gives the data to the smart contract.

[1] Currently, smart contracts can be programmed in others numerous languages, such as JavaScript, Go, Python, C #, Ruby, PHP, .and so on.

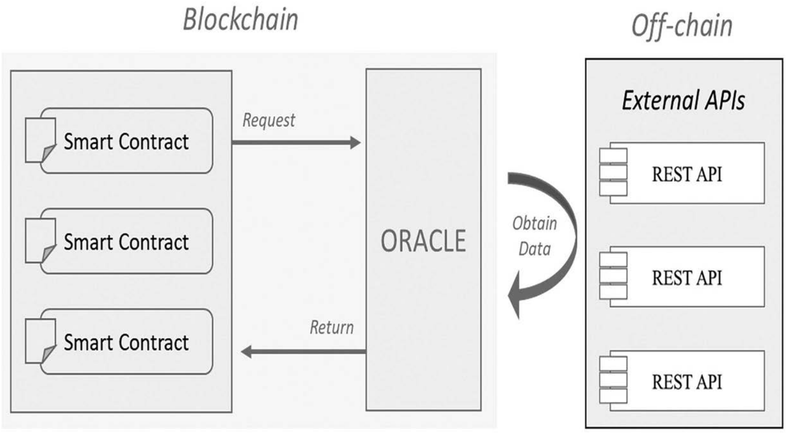


Figure 2 Blockchain with API server.

There are three oracle models: (a) **oracles coded** from scratch by and for a particular entity, (b) **centralized oracles**, or (c) **decentralized oracles**. The first of these options (Figure 2) is used to code an oracle from scratch for each use case. This method could lead to potential vulnerabilities and inefficiencies if it is not rigorously tested.

In a centralized oracle service, a third-party private company fetches and feeds data into the smart contract. But, when a centralized infrastructure is used for oracles, smart contracts lose their key features of being deterministic, reliable in their end-to-end execution and tamper-proof. On the other hand, a decentralized oracle network acts as both an oracle and a flexible framework for matching smart contract developers with secure and reliable oracle solutions.

Thus, oracle can be considered as a bridge between physical or real-life occurrences and blockchain based smart contract that retrieves, verifies, and passes the information to smart contracts for execution. It is important to note that smart contract must be invoked, which means that one must spend network resources for calling data from the outside world.

2.3 Software testing and early testing

Software testing is an essential part of Software Development Life Cycle (SDLC) [55] and without proper testing, the software should not be released. Specifically, the Software Testing Life Cycle (STLC) is a subset of the SDLC.

Traditionally, software testing has been a phase that always occurs at the end once coding phase is finished and before software is released. 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 the cost of refactoring or the cost of future maintenance activities.

This is clearly evidenced in the principles of software testing [24]. "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 3 shows an example of the Delayed Issue Effect (relating to the relative cost of fixing requirements issues at different phases of a project). In the Figure, it is clear that the impact of detecting a fail or inconsistence at the beginning (requirements phase) is quite reduced in comparation with detecting them in the last phases of the life cycle.

According to this chart, the cost to fix an error depends directly on which phase of the SDLC has been detected. Any error that is found may cause a domino effect [38]. An error that has not been found on time may require 100 times more efforts on its fixing after it gets to the stage of software

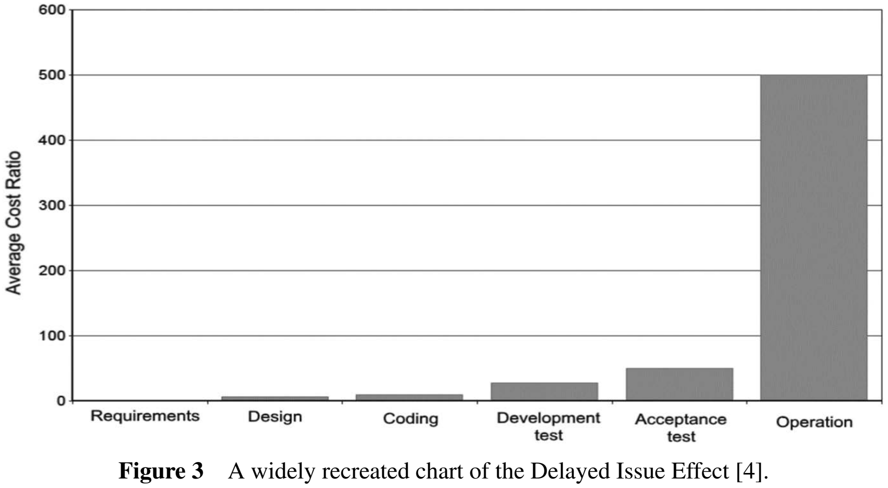


Figure 3 A widely recreated chart of the Delayed Issue Effect [4].

deployment. In this context, the requirements for the final software product are critical [38].

Maximum defects occur at the requirement phase, as noted in the "Inspecting Requirements" [50]: "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".

The book "Software Testing Techniques" [2] 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.

2.4 Model-based software development

Since a few years ago, modelling tools have helped to document the functionality of business processes and, through model transformations, to partially automate the generation of software source code. Unified Modelling Languages (UML) and other modelling standards are used for this purpose.

Models are typically easier to understand than software source code [18]. Therefore, their use allows improving development productivity and quality. It is easier to check the correctness of a model. In addition, modelling tools can ensure that the deployed code has not been modified after its generation from the model [30].

Compared to traditional software development (Figure 4), where the phases are clearly separated, model-based software development shows that the specification, design, and implementation phases have grown together much more strongly [9].

According to Seebache [41], model-based software development 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 also helping to develop solutions such as smart contracts.
- An architectural framework may be used to combine and transform different models and descriptive layers to facilitate the construction of global system.
- Based on a set of formalized meta-models, which in turn can be integrated and transformed into models with a higher degree of information, automation can be applied, and a smart contract code generated.

Other authors argue that the major advantages of a model-based design approach are [36]:

- The requirements are an integral part of the model and other parts of the model can be traced back to requirement.
- Models can be used to represent the desired behaviour of a system under test, or to represent testing strategies and a test environment.
- Automated system validation and verification reduce errors in the life cycle. Then automating the generation of quality code and automating testing ensures that the system implementation is correct and reliable.

Model-based testing is an application of model-based design for designing and optionally also executing artefacts to perform software testing or system testing. In other words, model-based testing is a technique where the runtime behaviour of an implementation under test is checked against predictions made by a formal specification, or model [37].

Therefore, in the context of blockchain-oriented applications, model-based software development is important for the following reasons [30]:

- 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 blockchain technology since they can be platform-agnostic, and model-based (model-based tools can be applied at multiple blockchain platforms).

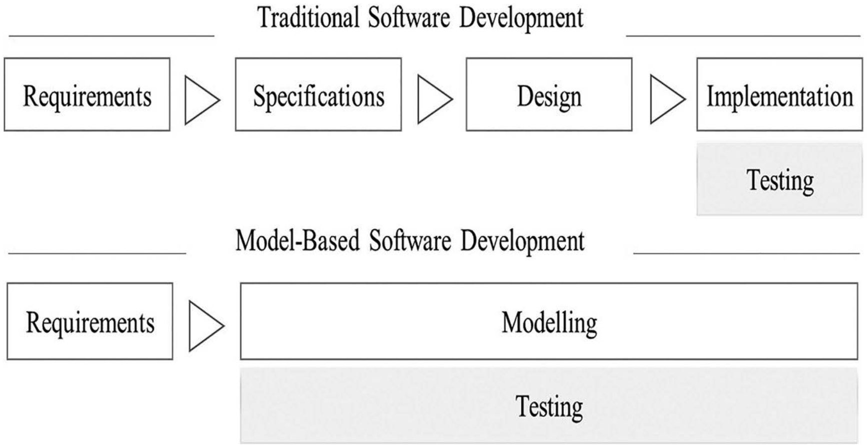


Figure 4 Traditional vs model-based software development [9].

- As discussed above, it is easier to check the correctness of a model and model-based tools can ensure that the deployed code has not been changed after its generation from the model.

3 Hypothesis

Blockchain is an immutable, transparent, and secure technology [45] for recording the state and ownership of an asset. This asset can be something tangible and diverse as IoT (internet of Things) devices, a piece of art, and so on. Even, it could be something intangible, for example intellectual property.

Nowadays, the blockchain smart contracts platforms[2] 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.[3]

That said, and given the immutability of blockchain technology, it is essential that, before deploying a smart contract code or an external API in a business network, this software goes through evaluation and validation processes. Because a defect of smart contract or external API could cause a non-repairable effect in blockchain network.

In order to achieve this objective, it is necessary that use case diagram, activity diagram for use case,[4] test case diagram, 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.

Given smart contracts have some specific characteristics, it would be necessary to introduce some new concepts in these diagrams, to better model and specify these smart contracts. These concepts can be introduced simply as UML stereotypes, which are tags that can be used in UML diagrams wherever needed. In some cases, it would be enough to introduce a specific notation such as the transfer of crypto coins in sequence or the activity diagram [31].

[2] Platforms as Ethereum (https://neo.org/), NEM (https://nem.io/), or NEO (https://neo.org/).

[3] 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). In this regards, a handful of companies, for example https://solidified.io/, reviewing the code and providing feedback on its quality and security.

[4] Use case diagrams are used to represent an overview of the functional requirements while activity diagrams provide a more detailed view.

The Software Development Life Cycle (SDLC) is a framework that defines the tasks to be performed at each step of the software development process. Therefore, 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 [47], streamlining the process of writing use cases, activity diagrams and automatic code generation, as well as the process of software testing in general and early testing in particular. It is crucial to start the software testing activity before starting the coding phase, it must be from the requirements and then refine the software testing in the analysis and design phases.

In this context, this hypothesis is raised: the development of model-based smart contract (and model-based APIs) could improve the functional verification and validation of the code by applying testing techniques from the early stages of the SDLC, which is known as early testing.

It should not be forgotten that functional and non-functional tests are required to validate and correct the behaviour of the smart contract before it is released into the blockchain network. In functional testing, all business rules, or requirements, previously collected, should be verified in various cases including valid / invalid arguments, boundary values, and argument combinations. But it is not possible to test blockchain successfully / efficiently without functional test automation. Due to the combinatorial explosion of many inputs and environmental factors in the test case preconditions, manually testing a contract is not only hard, but also inefficient. Given the features supported by blockchain networks, the focus should always be on the functionality, security, and performance of the blockchain network and smart contracts, although this paper focuses only on functionality.

4 Approach to Model-based Smart Contract Development and Testing

As mentioned above, model-based software development helps to describe and understand a system in detail. As shown in Figure 5, model-based software development can be applied to both on system modelling and test modelling (the products to be obtained are represented by grey boxes and the white boxes are the actions to be carried out to achieve this).

The detail shown in Figure 5 is discussed below.

Business rules or functional requirements, typically, are represented in a tabular format, which can facilitate the tracking of requirements throughout

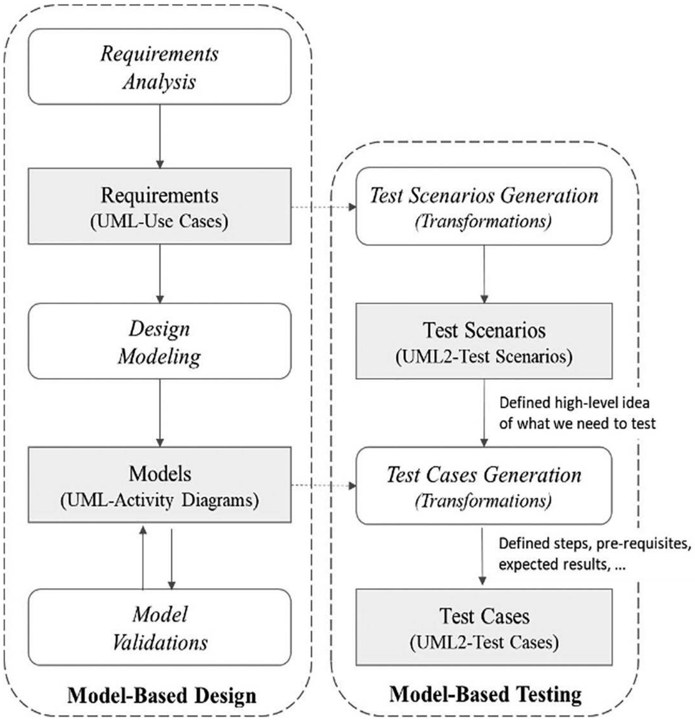

Requirements Analysis
Requirements (UML-Use Cases)
Design Modeling
Models (UML-Activity Diagrams)
Model Validations
Model-Based Design
Test Scenarios Generation (Transformations)
Test Scenarios (UML2-Test Scenarios)
Defined high-level idea of what we need to test
Test Cases Generation (Transformations)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

Figure 5 Model-based engineering.

the system's life cycle. It is important to know what happens when related requirements change or are dropped, which improves traceability.

The user case is a mean of capturing the requirements of systems, i.e., what systems are supposed to do [34]. The use cases focus on the documentation of the system's interaction with external actors (users of external systems). On the one hand, use case can be defined using text blocks or graphics artefacts. Text templates have been a classic tool for defining use case for many years. Several authors, such as Achour [1] or Cockburn [8] have proposed templates and text patterns for defining use case as text. On the other hand, UML is the standard for defining use case using graphics artefacts.

Therefore, UML defines a use case model to represent use cases and their relationships among actors and other use cases (see Figure 5). UML also introduces activity diagrams to define the specific behaviour of a use case. Several authors such as Escalona [14], or García-García [21] have proposed extensions to include in the models the same information than templates, so both artefacts can be used indistinctly.

Another popular artefact for requirements definition is based on user stories. A user story is a short text with a couple of sentences in the business language of the end-user of a software system. This story captures the essence of part of the work a user does or needs to do with the system. User stories do not define all the information for building the system. Instead, they are just a reminder of a future conversation with a product owner, or a user/customer [11].

User stories work mainly in agile processes because Agile Requirements Engineering practices [52] are focused on continuous interaction with customers in order to assure the evolution of requirements over time, and to enable requirements prioritization so that the most valuable functionalities are delivered first [42]. The main difference between user stories and the user case is that when a development team uses user stories, they obtain the information from users interactively during the development of the project. But, when a development team uses use case, they obtain the information in one step, storing that information in a text template or in a diagram [43].

Really, there is a good coverage of key functional requirements related to management of user stories, a taxonomy of user stories (like epics and spikes, which are two specific types of user stories with specific goals), release planning and sprint-by-sprint iteration planning [11].

Requirements have a common set of information regardless of the used technique: text templates, graphic diagrams, or user stories. All techniques share common objectives and can manage the same information. Choosing one or the other depends on the characteristics of the project or equipment and does not affect the quality of the generated software.

A basic example of UML use case diagrams is shown in Figure 6, which describes the relationships between use case and actors (and the relationship with blockchain network).

On the other hand, an illustrative example of a UML activity diagram is shown in Figure 7, which would describe the activities of one of the above use cases.

The use of UML is suggested because it is a visual language that supports the design and development of complex systems and, above all, because the most recent version 2.5.1 provides the means to describe a test model [1, 3]. Specifically, the Object Modelling Group (UML is supported by OMG) has defined a profile that is particularly suitable for testing. This profile, called UML Testing Profile, is a UML Profile dedicated to model-based testing. Its main goals are to design, visualize, specify, analyse, construct, and document the commonly used artefacts required for various testing approaches. UML Testing Profile 2.1 has improved the possibilities of various models related to test architecture [40]. This profile closes the gap between designers and testers, providing a means to use UML for both system modelling and test specification. With this profile, developers can reuse UML design documents for testing and allow the development of tests at an early stage of system development.

The UML testing profile fits perfectly into a model-based test environment. Therefore, any technique for generating test cases from a use case may be combined with UML Testing Profile, deriving test cases directly from UML-Activity diagrams.

In addition, UML is the most used software architecture specification language. According to Uzun et al. [46], 15 out of 31 software architects prefer UML. Second approach is 3 from 31 only. However, UML Testing Profile is only used in 5 out of 31 approaches. Despite this value, UML Testing Profile is the most used technique to represent test models in the 31 analysed approaches. This indicates that there is an interest in using UML in the testing process.

Therefore, model-based design can apply best practices and generate test cases from the early stages of the software development life cycle, thus reducing the number of coding errors. That is, the test scenarios are derived from the UML-Use case and the test cases are derived from the UML-Activity diagrams. This approach can also be used to check the consistency between the system execution traces and the behaviour of UML-Activity diagrams.

It is important to emphasize that the test cases are obtained through transformations, implementing all the transitions and all the criteria of the UML-Activity diagram of smart contract. They select the paths that go across a higher number of actions until all the actions of the activity diagrams have

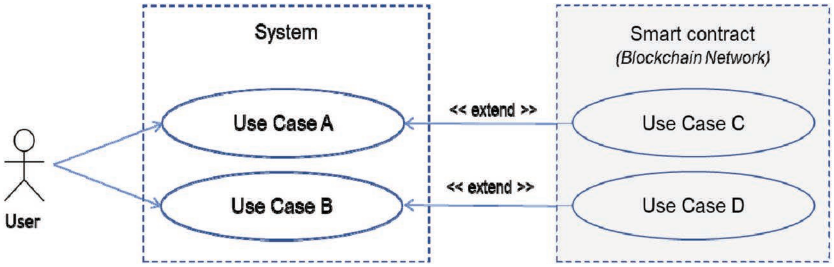


Figure 6 Example of a use case diagram for Blockchains context.

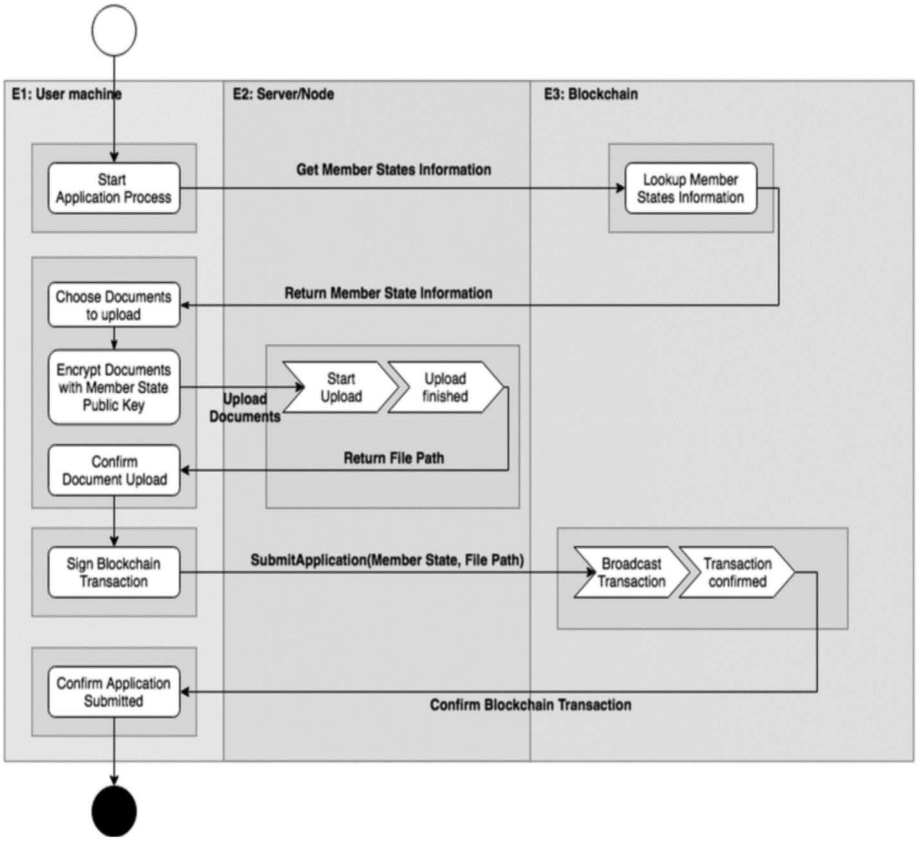

E1: User machine
E2: Server/Node
E3: Blockchain
Start Application Process
Get Member States Information
Lookup Member States Information
Return Member State Information
Choose Documents to upload
Encrypt Documents with Member State Public Key
Upload Documents
Start Upload
Upload finished
Return File Path
Confirm Document Upload
Sign Blockchain Transaction
SubmitApplication(Member State, File Path)
Broadcast Transaction
Transaction confirmed
Confirm Application Submitted
Confirm Blockchain Transaction

Figure 7 UML-Activity Diagram example of a use case [12].

been traversed at least once. For the all-transitions criteria, they select the path that traverse a higher number of object-flow edges until all of them have been crossed at least once.

An example of this process is described by Marchesi et al. [54]. They make a detailed proposal of the software development life cycle and its subsequent deployment in the blockchain network. The process uses formal notations, such as UML diagrams, which describe the system design, with additions to the specific representation concepts found in the development of blockchain.

Code generation is another important point to consider if model-based software development is applied. Smart contract code [16] and API code could be automatically generated, e.g., following the REST principles [17]

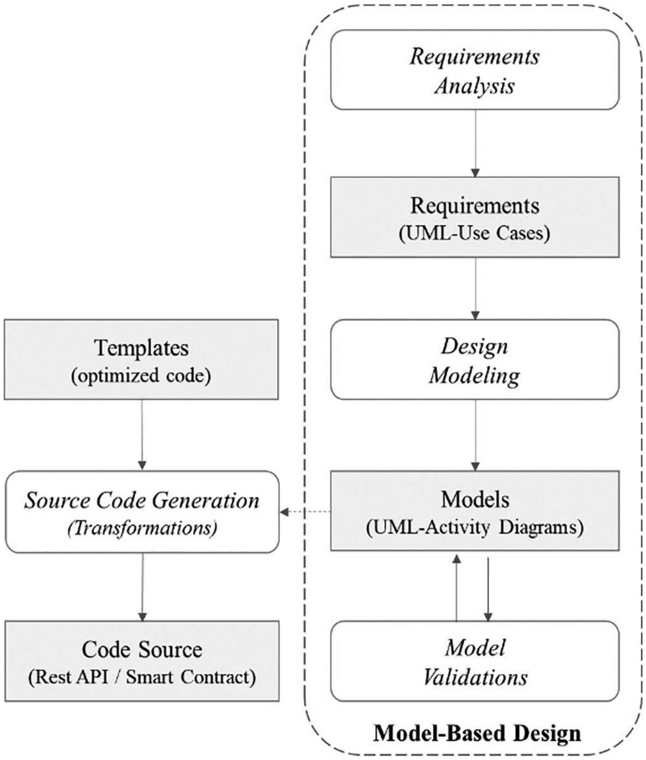

Requirements Analysis
Requirements (UML-Use Cases)
Design Modeling
Models (UML-Activity Diagrams)
Model Validations
Model-Based Design
Templates (optimized code)
Source Code Generation (Transformations)
Code Source (Rest API / Smart Contract)

Figure 8 Source code generation.

with different roles in blockchain as connectivity module, as security module, and so on [39].

As proposed in Figure 8, source code acquisition must be based on two fundamental elements: smart contract templates and the code generator from UML diagrams (the products to be obtained are represented by grey boxes and the white boxes are the actions to be carried out to achieve this).

The detail shown in Figure 8 is discussed below.

In Figure 8, one of the main elements they propose to use are smart contract templates.[5] These templates [7] can support legally enforceable

[5] The smart contract templates are based on the framework of Grigg's Ricardian Contract triple of "prose, parameters and code" [25, 26]. The main advantage of a Ricardian Contract is that if there is a dispute among parties involved, the case can be decided in court. This is not possible with smart contracts, which are only the instructions in which they are based on what is defined in an agreement. If something goes wrong, proving fraud or a scam in court is

smart contracts, using operational parameters to connect legal agreements to standardised code. The standardised code (the "smart contract code") would be derived from legal documentation (the "smart legal contract") that should make many or all the provisions of that contract.

An important issue is how to know if the smart contract code will correctly perform the provisions of the smart legal contract. Therefore, during development of smart contract code it is important to perform two steps: (i) Test the smart contract code to ensure that it is error-free, and (ii) Validate the behaviour of the smart contract code to ensure that it is faithful to the meaning of the contract. These smart contract templates have already demonstrated[6] a way to link standardised agreements to standardised code and so, in the near term, it may be possible to use with existing infrastructure [7].

Another element to highlight is the code generator. Some authors, such as Choudhury [6], supply a framework for the self-generation of smart contracts. This framework uses ontologies and semantic rules to encode domain-speci?c knowledge and then leverages the structure of abstract syntax trees to incorporate the required constraints. There are other initiatives, such as Lorikeet tool [48], in which the developed tool can automatically create smart contract code from specifications that are encoded in the business process and data registry models based on the transformations of the implemented models.

With these pillars, smart contracts (and external APIs) can be designed and tested using UML diagrams, and generate smart contract code from UML diagrams. In addition, round-trip engineering [23] could help keep the source code and smart contract design synchronized. Each time you generate code or update UML model, changes will be merged.

Code generation is also an important line of research and it is important to indicate that automatic code generation in a model-based development process [53] is vital to the cost effectiveness of development. It eliminates

difficult since a smart contract is not a legally binding agreement. Ricardian contracts do have a legal framework, and this adds clarity for all stakeholders. This means that, unlike smart contracts, lawyers are required to create and deploy a Ricardian contract.

[6] OpenBazaar (https://openbazaar.org/) was the first application to implement the Ricardian contracts. This open source software is a peer-to-peer e-commerce platform where you can trade anything directly with each other. This open source project uses Ricardian contract to check the legitimacy of an agreement between buyer and seller. Within the platform, they are called Trade Receipts, and it adds added security that all parties do the right thing. In case of fraud, there are legal records that can be used in court to settle the dispute. Apart from OpenBazaar, also the EOS blockchain network (https://eos.io/) is using Ricardian contracts to form an important part of any agreements made on the EOS blockchain. Ian Grigg is even a partner at block.one and they have big plans for further incorporating Ricardian contracts.

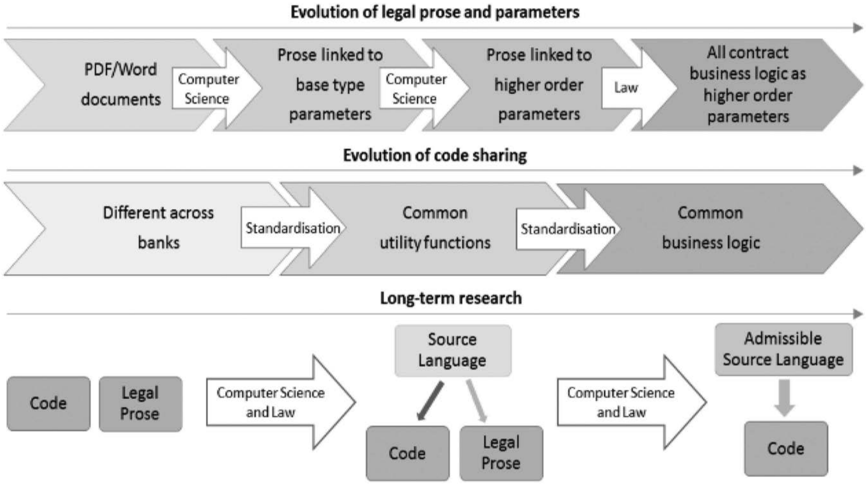


Figure 9 Potential evolution of important aspects of legally enforceable smart contracts [7].

the manual effort in coding from design, therefore, accelerating the process while decreasing the chance of errors when compared to manual coding from requirements or models. The generated code is designed according to the same principles, the nomenclature rule matches, then the code will always work as expected. The quality of the code is consistent. With manually written code, however, different software developers may have different styles and occasionally introduce errors in even the most repetitive code. Consequently, automation will reduce the level of inherent complexity associated with smart contracts and encourage their rapid adoption in different sectors such as government, supply chain, entertainment, among others.

5 Conclusions And Future Work

This paper has presented a discussion about the technological constraints and the current situation of the smart contract.

Testing of smart contracts can be achieved using model-based development. However, some questions arise. On the one hand, we introduce a preliminary view of an approach based on the use of models and the necessity of having a framework for orchestrating the smart contract Development Life Cycle. On the other hand, the benefits of early testing design are highlighted, among them, the detection of a high percentage of defects before coding/implementation begins, the verification of inconsistencies and omissions in the test base by increasing the test quality. This would help prevent defects from being introduced into the code. since the later in the lifecycle defects are detected, the more expensive they are to fix (It is more cost effective to find and repair defects from requirements stage). In model-based smart contract design, software engineers use models to develop their designs from the functional requirements or business rules. In addition, automatic code generation from the detailed design models removes the chance of introducing translation errors going from the design to the code.

A sound software engineering approach might greatly help in overcoming many of the issues plaguing blockchain development, providing software engineer with instruments similar to those typically used in traditional software engineering to deal with architectural design, security issues, testing planes and strategies to improve software quality and maintenance [31].

In the same way, model-based design enables the use of verification as a parallel activity that occurs throughout the development process. Doing test and verification along every step of the SDLC means finding errors at their point of introduction.

Model-based testing allows evaluation and validation processes before having the smart contract executable code. It can be ensured with a high percentage that the code is consistent with their models by applying code generation from these models. Developing tests in parallel of design and development work allows early detection of potential problems, and it significantly reduces the cost and time required for fixing them. By thinking about testing while you develop the model, you will design better for testability, thus ensuring that the design is fully testable. Tests and requirements have a synergistic relationship and early testing helps to find missing functional requirements. Model-based testing facilitates traceability of what you did; it is possible to show someone that you met his needs; and it is possible to repeat your results.

In previous sections, we have exposed that there are tools that support the generation of models and code for smart contracts. However, time and training are needed to use these tools. In addition, although the tools are based on widely recognized standards such as UML, it is not easy to change one tool for another due to the details necessary to perform automatic generation. Some details are the use of specific UML profiles or the use of extended notation. In our experience, the use of model-based testing tools saves a lot of repetitive and error-prone work - but they also limit the flexibility of the work that can be done -, because you have to work within the limits of what the tool can do.

Future work has been identified to extend smart contract test using model-based testing. The first one is to study the current situation of source code generator tools and early testing in blockchain network. To this end, a Systematic Literature Review (SLR) is being developed following the approach of Kitchenhand [28]. 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 (Figures 5 and 8) 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, 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 pre-tests for their collaboration.

This paper is an extended version of article "Towards an approach for applying early testing to smart contracts", presented at the 19th International Conference on Web Information Systems and Technologies – WEBIST 2019.

This research has also been supported by POLOLAS Project (TIN2016-76956-C3-2-R) of the Spanish the Ministry of Economy and Competitiveness and SocietySoft Project (AT17_5904_USE) of the Regional Ministry of Economy, Knowledge, Research and University of the Andalusian Government.

References

[1] Achour, C.B. (1998). Writing and Correcting Textual Smart Contract for System Design. Natural Language and Information Systems Workshop. Vienna, Austria.

[2] Beizer, B. (1990). Software Testing Techniques. Van Nostrand Reinhold Company Limited.

[3] Binder, R. V. (2000). Testing Object-Oriented Systems. Addison-Wesley. USA.

[4] Boehm B. (1981). Software Engineering Economics. Prentice Hall, Englewood Cliffs, NJ.

[5] Buterin, V. (2014). Ethereum: A next-generation Smart Contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Pape.

[6] Choudhury, O. Rudolph, N., Sylla, I., Fairoza, N. (2018). Auto-Generation of Smart Contracts from Domain-Specific Ontologies and Semantic Rules. 10.1109/Cybermatics_2018.2018.00183.

[7] Clack, C.D., Bakshi, V.A., Braine, L. (2016). Smart Contract Templates: essential requirements and design options. https://arxiv.org/pdf/1612.04496.pdf

[8] Cockburn, A. (2000). Writing Effective Use Cases. Addison-Wesley 1st edition. USA.

[9] Conrad, M., Fey, I., Sadeghipour, S. (2005). Systematic Model-Based Testing of Embedded Automotive Software. Electronic Notes in Theoretical Computer Smart Contract (Book).

[10] 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).

[11] Dimitrijević, S., Jovanovic, J., Devedžić, V. (2015). A comparative study of software tools for user story management. Information and Software Technology, 57 (1), 352–368. https://doi.org/10.1016/j.infsof.2014.05.012

[12] Liss, F. (2018). Blockchain and the EU ETS: An architecture and a prototype of a decentralized emission trading system based on smart contracts. https://doi.org/10.13140/RG.2.2.15751.65448

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

[14] 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).

[15] 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.

[16] Fielding, R. T., Taylor R.N. (2000). Architectural styles and the design of network-based software architectures. Doctoral Dissertation. Architectural styles and the design of network-based software architectures. University of California, Irvine © 2000. ISBN:0-599-87118-0

[17] Fielding, R. T., Taylor R.N. (2002). Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology, Vol. 2,

[18] 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

[19] 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.

[20] García-García, J. A., Enríquez, J. G., García-Borgoñón, L., Arévalo, C., & Morillo, E. (2017). An 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.

[21] 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.

[22] Chainlink (2019). Interoperability and Connectivity: Unlocking Smart Contracts 3.0. https://blog.chain.link/interoperability-and-connectivity-unlocking-smart-contracts-3-0-2

[23] Pham, V.C. (2018). Model-Based Software Engineering: Methodologies for Model – Code Synchronization in Reactive System Development. Embedded Systems. Université Paris – Saclay.

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

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

[26] 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.

[27] 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.

[28] Kitchenham B., Brereton P. (2013). A systematic review of systematic review process research in software engineering. Information & Software Technology.

[29] Kundu, D., Samanta, D., Mall, R. (2013). Automatic code generation from unified modelling language sequence diagrams. Software, IET. 7. 12-28. 10.1049/iet-sen.2011.0080.

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

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

[32] Nakamoto, S. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System.

[33] Nofer, M., Gomber, P., Hinz, O., Schiereck, D. (2017). Blockchain. Business & Information Systems Engineering. June 2017, Volume 59, Issue 3, pp. 183–187.

[34] OMG. (2017). An OMG ®Unified Modeling Language ®Publication OMG ®Unified Modeling Language ®(OMG UML ®) OMG Document Number: Date. (December), 796. https://www.omg.org/spec/UML/20161101/PrimitiveTypes.xmi

[35] Omohundro, S. (2014). Cryptocurrencies, Smart Contracts, and artificial intelligence. Published in AI Matters.

[36] PivotPoint Technology^{TM}. (2019). Digital Engineering Solutions? for Wicked Problems (https://Pivotpt.com)

[37] Pretschner, A., Prenninger, W., Wagner, S., Kuhnel, C., Baumgartner, M., Sostawa, B., Z Íolch, R., Stauner, T. (2005). One evaluation of model-based testing and its automation. ICSE'05.

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

[39] Sandoval, K. (2018). The Role of APIs In Blockchain. https://nordicapis.com/the-role-of-apis-in-Blockchain/. Bloc Nordic APIs.

[40] Satoh, A., Ban, S., Harayama, Y., Yamamoto, K. (2019). Designing fulfilling test cases with test aspect model. Proceedings - 2019 IEEE 12th International Conference on Software Testing, Verification and Validation Workshops, ICSTW2019, 153–158. https://doi.org/10.1109/ICSTW.2019.00044

[41] Seebacher, S., Maleshkova, M. (2018). Model-driven Approach for the Description of Blockchain Business Networks. Proceedings of the 51st Hawaii International Conference on System Smart Contract.

[42] Sillitti, A., Succi, G. (2005). Requirements Engineering for Agile Methods. In: Aurum A., Wohlin C. (eds) Engineering and Managing Software Requirements. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/3-540-28244-0_14

[43] Sillitti, A., Ceschi, M., Russo, B., Succi, G. (2005) Managing Uncertainty in Requirements: A Survey in Documentation-driven and Agile Companies, in: 11th IEEE International Software Metrics Symposium (METRICS '05). DOI: 10.1109/METRICS.2005.29

[44] Staples, M., Chen, S., Falamaki, S., Ponomarev, A., Rimba, P. Tran, A. B., Weber, I. Xu, X., Zhu, L. (2017). Risks and opportunities for systems using Blockchain and Smart Contracts. Technical Report. Data61 (CSIRO), Sydney.

[45] Sultan, K., Ruhi, U., Lakhani, R. (2018). Conceptualizing Blockchains: Characteristics & Applications. 11th IADIS International Conference Information Systems.

[46] Uzun, B., Tekinerdogan, B. (2018). Model-driven architecture-based testing: A systematic literature review. Information and Software Technology, 102 (May), 30–48. https://doi.org/10.1016/j.infsof.2018.05.004

[47] The Modex Team (2019). The life cycle of Smart Contract development https://blog.modex.tech/the-life-cycle-of-smart-contract-development-58b04f65de09

[48] Tran, A.B., 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.

[49] Walport, M. (2016). Distributed Ledger Technology: Beyond Blockchain. A report by the UK Government Chief Smart Scientific Adviser. https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf

[50] Wiegers, K. E. (2001). Inspecting Requirements. StickyMinds.com Weekly Column.

[51] Szabo, N. (1997). Formalizing and Securing Relationships on Public Networks. First Monday, 2, No. 9. http://firstmonday.org/ojs/index.php/fm/article/view/548/469

[52] Schön E-M., Winter, D., Escalona, M.J., Thomaschewski, J. (2017). Key Challenges in Agile Requirements Engineering. 18th International Conference on Agile Software Development (XP2017), At Cologne.

[53] Bialy, M., Pantelic, V., Jaskolka, J., Schaap, A., Patcas, L., Lawford, M. Wassyng, A. (2017). Software Engineering for Model-Based Development by Domain Experts. Handbook of System Safety and Security. https://doi.org/10.1016/B978-0-12-803773-7.00003-6

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

[55] Uddin, A.; Anand, A. (2019). Importance of Software Testing in the Process of Software Development. IJSRD – International Journal for Scientific Research & Development. Vol. 6, Issue 12, 201.

Biographies

N. Sánchez-Gómez has developed a large part of its professional career in the technology and process consultancy sector, both in the private and public sectors. Throughout more than thirty years of professional experience, he has gone from implementing ICT solutions to supervising work teams, managing clients and leading ICT projects. He is currently a member of the Web Engineering and Early Testing Research Group. In recent years, he has been coordinating different projects of the research group, including the Project Management Office of the Ministry of Culture (Andalusian Regional Government).

He currently has a broad knowledge of the functions and processes that make up the activity environment of the sectors in which he has participated and has completed his studies in Computer Engineering, with the Degree in Computer Engineering and the University Master in Engineering and Software Technology, both at the University of Seville, with the knowledge and skills of people management, ICT project management, customer management and practical application of computer engineering methodologies and techniques, in addition to obtaining the Prince2® Foundation Certified, the ISTQB® Certified Tester, Foundation Level and the PMP® Certified.

Previously, from 2001 to 2009, he developed his professional activity as Manager of the company everis Spain, being responsible for different accounts in both the public and private sectors. From 1990 to 2001, he worked for the company Coritel (Accenture group), where he also carried out management and project management activities. Further information about her research activities and her list of publications can be found at https://investigacion.us.es/sisius/sis_showpub.php?idpers=20733

L. Morales-Trujillo Graduated in Health Engineering since 2016 and Master in Software Engineering and Technology since 2018; both by the University of Seville (US).Since 2016 linked to the Web Engineering and Early Testing (IWT2) research group, belonging to the department. Languages and Computer Systems of the Higher Technical School of Computer Engineering of the US. Currently, enrolled in the doctorate program in Computer Engineering at the US, with a line of research according to my academic training: Ensuring the traceability of elements in a context of systems of systems. In addition, linked to the US within the Recruitment Plan for Young Research Personnel and R&D Support Technicians, Framework of the National Youth Guarantee System of the 2014–2020 Youth Employment Operational Program. Being part of a research group has allowed me to get to know different R&D lines, attend conferences, participate in R&D contracts in collaboration with companies, practice as a visiting professor, as well as continue my academic studies.

The lines of research in which I have worked are: Optimized and Flexible Management of Business Processes, Early Testing and Hybrid Simulation Models in Software Production, Guided Solutions to Systematize Early Quality Assurance of Software and Guided Mechanisms in Stages Early for

Software Improvement. I have practiced (2019) as a guest professor in the subject “Modeling and Analysis of Requirements in Information Systems” of the Bachelor’s Degree in Computer Engineering in Computer Technologies, teaching requirements validation activities in the context of developing an information system to support the clinical services management. I have done a 6-month pre-doctoral stay at the Universidad a Distancia de Madrid, in the course 18/19. Further information about her research activities and her list of publications can be found at https://www.investigacion.us.es/sisius/sis_showpub.php?idpers=25277

J. J. Gutiérrez was awarded his PhD in Computer Science by the University of Seville, Spain, in 2011. Since 2004 he has been a professor in the Department of Computer Languages and Systems at the University of Seville and since 2006 he has been a collaborating professor.

He is currently a member of the Web Engineering and Early Testing Research Group. Among his most notable research results, it is worth mentioning his transfer to the business world. With the development of the concept of early testing and its integration with the NDT methodology also developed within the research group, he has managed to develop a set of methodological solutions for the development and quality assurance that has been widely used in the Andalusian and national business network or even by international companies. This can be measured not only in the transfer projects, but also in the number of publications with companies and in the tools registered. Further information about her research activities and her list of publications can be found at https://investigacion.us.es/sisius/sis_showpub.php?idpers=11730

J. Torres-Valderrama received his MSc and the Phd in Computer Systems in 1997. He has been working in the Department of Computer Languages and Systems at the Seville's University since 1991, where he is currently a senior lecturer. Her main research interests are related to requirements engineering, web-based systems development, user interfaces, usability and Early Software Testing. In these areas, she has directed several PhD theses and published numerous papers in journals and congresses. He has managed and participated in a high number of projects related to her areas of research.

He has been dean of School of Computer Engineering at Seville's University from 2006 to 2014 and he is currently manager of the Foundation for Research and Development of Information Technology in Andalusia since 2016. Further information about her research activities and her list of publications can be found at https://investigacion.us.es/sisius/sis_showpub.php?idpers=3278