Information Systems and e-Business Management (2021) 19:863–882
https://doi.org/10.1007/s10257-020-00465-5

ORIGINAL ARTICLE

Laying the foundation for smart contract development: an integrated engineering process model

Christian Sillaber[1] · Bernhard Waltl[2] · Horst Treiblmaier[3] · Ulrich Gallersdörfer[2] · Michael Felderer[1,4]

Received: 25 February 2019 / Revised: 12 October 2019 / Accepted: 18 January 2020 /
Published online: 6 February 2020

Abstract

Smart contracts are seen as the major building blocks for future autonomous blockchain- and Distributed Ledger Technology (DLT)-based applications. Engineering such contracts for trustless, append-only, and decentralized digital ledgers allows mutually distrustful parties to transform legal requirements into immutable and formalized rules. Previous experience shows this to be a challenging task due to demanding socio-technical ecosystems and the specificities of decentralized ledger technology. In this paper, we therefore develop an integrated process model for engineering DLT-based smart contracts that accounts for the specificities of DLT. This model was iteratively refined with the support of industry experts. The model explicitly accounts for the immutability of the trustless, append-only, and decentralized DLT ecosystem, and thereby overcomes certain limitations of traditional software engineering process models. More specifically, it consists of five successive and closely intertwined phases: conceptualization, implementation, approval, execution, and finalization. For each phase, the respective activities, roles, and artifacts are identified and discussed in detail. Applying such a model when engineering smart contracts will help software engineers and developers to better understand and streamline the engineering process of DLTs in general and blockchain in particular. Furthermore, this model serves as a generic framework which will support application development in all fields in which DLT can be applied.

Keywords Smart contract · Development process model · Software engineering · Blockchain · Distributed ledger technology · Survey · Design science · Trustless append-only decentralized digital ledgers (TADDL)

✉ Michael Felderer
michael.felderer@uibk.ac.at

Extended author information available on the last page of the article

1 Introduction

Within the last couple of years, blockchain technology, or, more generally, distributed ledger technology (DLT), has become a highly popular research topic and been recognized as a potential game-changer for the industry. The pseudonymous author (or group of authors) Satoshi Nakamoto, who mentioned neither the term "blockchain" nor "DLT" in his paper, introduced the cryptocurrency Bitcoin as the first use case of this technology (Nakamoto 2008). The years following this seminal paper were characterized by intense discussions in academic communities specialized in cryptography and dedicated online groups, but the full potential of the technology was not yet properly understood within applied academic and business-oriented communities. This situation changed around the year 2014 when the exchange rate of Bitcoin started to soar and several authors highlighted the potential of the technology for countless use cases beyond cryptocurrencies (Swan 2015; Tapscott and Tapscott 2016). Not surprisingly, various academic communities have now started to rigorously investigate the topic and an ever-increasing number of papers is being published in various academic outlets (Johansen 2018).

The immutable, trustless model of decentralized computation and transaction handling which is provided by the blockchain strives to ensure fairness for all participating actors and has led to an ever-increasing market capitalization of cryptocurrencies in recent years. By June 2018, the capitalization of the five leading cryptocurrencies (Bitcoin, Ethereum, Ripple, Bitcoin Cash, and Litecoin) exceeded 193 billion dollars (Coinmarketcap 2018). However, the functionality of blockchain does not stop at cryptocurrencies. One of the most widely discussed features of blockchain technology is the possibility to create decentralized and self-executing programs: so-called *smart contracts.* Current application scenarios include areas as diverse as supply chain/logistics, finance, tourism, Internet of Things (IoT), project operations, gamification and education (Treiblmaier and Zeinzinger 2018; Tapscott and Tapscott 2016). Numerous organizations are already applying this technology stack to supplement or supplant existing legal and financial transactions (Egelund-Müller et al. 2017; Fanning and Centers2016; Friedlmaier et al. 2016; Clack et al. 2016; Notheisen et al. 2017). The huge monetary values processed through smart contracts and/or represented by cryptocurrencies necessitate structured software development processes and high levels of quality assurance. Incidents such as the DAO attack (Siegel 2016) as well as the King of the Ether incident (KingOfTheEther 2016), aggregating damage in excess of 60 million dollars, illustrate (1) that the current ad-hoc style of engineering is not suitable for such high-value transactions, (2) that existing software engineering approaches do not ensure a sufficient level software quality, and (3) that these approaches are either unsuitable for or misaligned with the specificities of blockchain technology (Atzei et al. 2016). Atzei et al. (2017, p. 182) point out that "a common cause of insecurity of smart contracts is the difficulty of detecting mismatches between their intended behavior and the actual one". Subsequently, Destefanis et al. (2018) encouraged the academic community to further investigate and develop blockchain-oriented software engineering processes; We answer that call.

Traditional software engineering focuses on principles for developing high-quality software systems and maintaining the systems as they evolve in real-world environments (Mens et al. 2010; Edan and Pliskin 2001). Software that does not evolve during its lifetime, however, will not be able to keep up with changing requirements and will become outdated over time. This has profound implications for existing software process models, which must respond to the increasing need for change and evolution by introducing iterative, incremental, and evolutionary approaches (Mens et al. 2010; Boehm and Turner 2005; Beck et al. 2001). In demonstration, software maintenance and evolution have emerged in the last decade as key research fields that explicitly differentiate between the time phases before, during, and after the software is delivered: denoted development time, deployment time, and runtime, respectively (Jacobson et al. 1999). Post-deployment changes are typically realized by returning to regular development activities, which eventually result in a new version of a software product or a patch that is released to replace or enhance the currently running version during scheduled downtimes. This process is structured by change management activities (Stark 2015; Rajlich and Bennett 2000; Bennett and Rajlich 2000). All such changes that occur after the initial development time are impossible, however, in settings where the contracts are published on blockchain and are immutable from that point onward. To clarify, and to counter an often-repeated yet incorrect narrative, DLT systems are only immutable with respect to their specific context and rules. Changes in the execution environment, in the stakeholder agreement (e.g., switching to another technology or invalidating specific entries), or in the usage of specific patterns (e.g., proxy pattern) yield mutability.

In this paper, we systematically develop a smart contract engineering process that clearly outlines its respective elements and artifacts. This process description represents an essential tool for numerous strategic and operational activities since it helps in defining priorities, clarifying risks, and managing expectations and time frames. We further develop a framework that supports stakeholders of smart contract engineering processes in their coordination efforts and which can be used for activities such as project management and legal risk management, complexity and standards management, as well as for security and quality management.

This paper is structured as follows: First, we identify research topics related to DLT technology and give a short introduction to trustless, append-only, decentralized digital ledgers, and to related software engineering process models. Second, we describe our methodological approach consisting of expert interviews and qualitative content analysis. Third, we develop an integrated process model for DLT technology in a stepwise process. Finally, a short discussion and a comparison to conventional engineering processes concludes this paper.

2 Identification of DLT research topics

The immutable nature of smart contracts in trustless, append-only, and decentralized digital ledgers makes the traditional software engineering lifecycle both inappropriate and insufficient (Sillaber and Waltl 2017). Instead, the technical specificities of blockchain technologies demand central consideration as a robust frame of reference

that helps in decomposing its overall complexity and accommodating the new requirements of smart contract engineering. In Table 1 we list several research topics and questions pertaining to a variety of different managerial aspects of software development that need to be systematically addressed. In the domain of *project and legal risk management*, we list topics surrounding the consideration of all relevant issues and requirements prior to the deployment of the smart contract, including cost comparisons, project duration, mitigation of legal and project-related risks, and potential problems arising due to legal requirements related to "anti-money laundering" (AML) and "know your customer" (KYC). When it comes to *complexity and standards management*, questions arise as to which concepts and elements of smarts contracts need to be standardized, which components of contracts should be individualized: considerations include the decomposition of complexity, the impact of smart contracts on negotiation processes, the integration into legacy systems as well as the portability of existing systems on the blockchain, the identification of the best modeling languages and smart contract patterns, and the handling of contracts that are interrelated. Important issues related to *security and quality management* include the testing and validation of smart contracts, the mitigation of risks that might arise from bugs and vulnerabilities not known at the time of development, the auditing of smart contracts, and the security of the underlying platform.

3 Related work

Although the term *blockchain* is relatively new (Swan 2015; Tapscott and Tapscott 2016), its underlying concepts are not. Some of the foundations of this technology, such as Merkle trees, proof of work algorithms, or smart contracts, were already developed decades ago (Narayanan and Clark 2017). Smart contract engineering, therefore, builds on (1) the conceptual foundation of smart contracts, as well as on (2) state-of-the-art software engineering with a focus on blockchain technology. We briefly outline both topics in the following sections.

3.1 Smart contracts in trustless, append-only, decentralized digital ledgers

The term *smart contract* was introduced by Szabo (1997) when he first described how the computer-based execution of contracts between two parties can be secured without requiring a third party for intermediation or confirmation. His original article provides the first description of decentralized smart contracts as computer programs that are executed by all participants. This allows all participating parties, who do not necessarily know or trust each other, to securely transact with each other. The correct execution of these programs is ensured by a so-called consensus protocol (Luu et al. 2016).

The basic technological properties of trustless, append-only, decentralized digital ledgers (TADDL), which includes blockchain technology, are well-studied and described in the literature (e.g., Tschorsch and Scheuermann 2016). Several separate active research streams focusing on specific technological issues of TADDLs can

Table 1 Research topics

Domain	Related questions	Sources
Project and legal risk management	How can all relevant issues and requirements of stakeholders be safeguarded before the smart contract is finalized in the blockchain? What are the costs of smart contracts compared to offline contracts? What is the expected project duration of the implementation? How can well-known risks be mitigated and unknown risks be identified as early as possible? How can legal risks be mitigated? How can challenges from AML and KYC be efficiently addressed?	Wang et al. (2016), Rückeshäuser (2017), Deshpande et al. (2017), Porru et al. (2017), Xu et al. (2017), Böhme et al. (2015), Pesch and Sillaber (2017), Fairfield (2014), Kiviat (2015), Moyano and Ross (2017)
Complexity and standards management	Which concepts and elements of smart contracts need to be standardized across the entire ecosystem or within specific legal environments? Which components of smart contracts should be individualized for each customer? How can the complexity that results from the holistic and comprehensive nature of legal requirements and their translation into smart contracts be decomposed to make it manageable? How can the codification of legal requirements into smart contract code improve the negotiation process between the involved parties? How can smart contracts be integrated into existing IT systems? Which parts of an enterprise IT Architecture can be ported to blockchain? What are the best modeling notations for smart contract development? What are the patterns in smart contracts? How should contracts that have dependencies between them be dealt with?	Seijas et al. (2016), Frantz and Nowostawski (2016), Marjanovic and Milosevic (2001), Clack et al. (2016), Beck and Müller-Bloch (2017), Bartoletti and Pompianu (2017)
Security and quality management	How can smart contracts be tested and validated? How can stakeholders efficiently mitigate risks from bugs and vulnerabilities in smart contracts? How can smart contracts be audited? How secure is the underlying platform?	Delmolino et al. (2016), Clack et al. (2016), Bhargavan et al. (2016), Atzei et al. (2016), Idelberger et al. (2016), Leitner et al. (2007)

be identified, including topics such as anonymity versus pseudonymity, transaction rates, and proof-of-X (Tschorsch and Scheuermann 2016; Anderson et al. 2016). A TADDL is a decentralized virtual state and computing machine that enables several parties to share a common state, the integrity of which is ensured and verified by other participating parties or "volunteers", including, for example, miners (Anderson et al. 2016). Various incentives promote participation in the mining process, most

notably coin rewards. The use of a so-called consensus protocol that is binding for all participating parties and which forms the mechanism through which consensus is achieved within a peer-to-peer network, implies that the data being stored—cryptocurrency asset balances, for instance—are accepted by all participants. With the components provided by the TADDL, complex digital asset transactions and financial instruments can be created (Buterin 2014; Koulu 2016; Anderson et al. 2016). The use of smart contracts executed in a TADDL can be observed in many different domains, ranging from online gambling to fundraising (e.g., Porru et al. 2017; Egelund-Müller et al. 2017; Xu et al. 2017; Böhme et al. 2015; Klöhn et al. 2018).

3.2 Software engineering process models

The IEEE 1074-1995 Standard for Developing Software Lifecycle Processes defines a process as a set of steps that can be executed in a certain predefined, sequential, parallel, or conditional order (IEEE 1995). Software engineering processes are part of the general Software Engineering Body of Knowledge (Bourque and Fairley 2014). Various process models cover the order and frequency of phases in software projects. Those phases typically include planning, analysis, design, implementation, testing, and maintenance. Waterfall models progress sequentially through these phases, whereas iterative models are typified by repeated execution of the waterfall phases, in whole or in part (Braude and Bernstein 2016). Differing from these phase-oriented process models, agile process models are based on the principles of individuals and interaction, working software, customer collaboration, and fast response to change (Beck et al. 2001; Vidgen and Wang 2009; Lee and Xia 2010). A recent trend is to combine phase-oriented with agile process models to obtain hybrid software engineering process models (Kuhrmann et al. 2017).

Modern software engineering approaches rely heavily on the (re-)use of software patterns (Kuhrmann et al. 2017). Patterns are collections of abstract best practices of software code that engineers can easily adapt. These best practices are the result of previous software engineering experience and often allow faster, more secure, and more reliable software development. As industry experience with smart contracts grows, it is very likely that a set of smart contract patterns will emerge in order to foster efficiency and effectiveness in the creation of smart contracts.

In their overview on the current status of research and practice regarding software engineering process models, Fuggetta and Di Nitto (2014) highlight several challenges caused by the Internet as a basic development, execution, distribution, and business infrastructure. They list research issues such as the fading distinction between design, development, and operation, but also highlight topics such as security, privacy, and trust. Blockchain-oriented software engineering has also attracted recent interest. Porru et al. (2017), for example, outline new research directions for blockchain-oriented software engineering processes, which include the areas of collaboration, enhancement of testing and debugging, as well as the creation of software tools for smart contract languages. In this paper, we extend previous research by developing an integrated process model for smart contract engineering.

4 Methodology

We conducted interviews on smart contracts with eleven industry experts. Table 2 gives an overview of the participants, their organizational roles, qualifications, and previous involvement in blockchain projects, structured by blockchain type and use case. The primary goal of the interviews was to get a better understanding of how the study participants develop smart contracts and which processes, artifacts, and tools they apply. We used a Delphi study approach wherein the findings from the first round (interview partners 1–9) were evaluated and refined in the second round, in which interview partners 8–11 participated (Prusty et al. 2017). Interview partners 8 and 9 were part of both rounds and helped to connect the findings by critically commenting on the feedback from the second round. The experts were identified by contacting the respective leaders of the development teams of the 20 largest blockchain projects, as measured by their token market capitalization listed on ICOAlert (www.icoalert.com) as at November 2017, as well as through the authors' personal networks. All potential experts were invited via email and eleven of them agreed to participate in our study. The participants were briefly informed by the researchers about the context, goals, and scope of the study, and the interviews were conducted via video conferencing or in person. Each interview was recorded and transcribed. After the interviews, the resulting process model, as well as the changes resulting from the interview, were sent to the interviewees for further feedback, which was then again incorporated by the researchers.

We used the open questions shown in Table 3 to structure the interviews and frequently applied follow-up questions to clarify specific issues (cf., Chau and Tam 1997). In a first step, the experts were informed about the goals and the procedures of this research project. Most notably, we presented various intermediate versions of the process model that included modifications and extensions based on the findings generated from previous interviews. Next, they discussed the different stages of the processes they use in their companies in a stepwise manner and included their findings in the model. Finally, summaries were produced from the interviews in order to derive concepts and constructs for further model development (Mayring 2014; Lacity and Janson 1994).

5 Process model development

In the following sections we develop the smart contract engineering process in a stepwise manner. First, we discuss the conceptual base and describe the main types of artifacts that emerged from the interviews. Second, we discuss the findings from the qualitative interviews with several software developers. Third, we present various roles, activities, and artifacts and, fourth, we incorporate these components into one integrative model.

5.1 Conceptual base

We followed a design science approach to precisely define the respective steps of the process model (Baskerville et al. 2018; Zakarian and Kusiak 2001). The core concept of design science is the artifact: an object that can be instantiated with physical or social properties. Examples of artifacts can be as diverse as software, models, or norms (Hevner et al. 2004). In their proposed research framework for the conceptualization of design science research within information systems (IS) research, Hevner et al. (2004) propose three integrated dimensions: (1) the environment including people, organizations, and technology, (2) IS research pinpointing the creation and justification of artifacts, and (3) the knowledge base bringing forward foundations and methodologies to be used in the creation and evaluation of artifacts. More specifically, March and Smith (1995) differentiate between four types of artifacts: constructs, models, methods, and instantiations. *Constructs*, which consist of language and vocabulary specifying problems and solutions, form the baseline design science vocabulary. The construct level establishes a common understanding of the involved entities, by identifying those entities, their attributes, and the relationships between them. Furthermore, constructs describe the terms being used and ensure their consistent usage throughout the domain. *Models* are descriptions and representations of real-world phenomena with a focus on utility, not truth (March and Smith 1995). They can therefore be abstract and may represent nothing more than arbitrary aggregations and groupings of instances. In order to be useful, the entities chosen for the model have to be representative of the underlying information system (Wood 2014). The steps needed to execute a specific process are called *methods*, which are procedures for solving problems and developing solutions. Methods are built on constructs and models. They are used to transform constructs and models from one representation into another and consequently operate on models and concepts as input and output parameters. Methods also subsume abstract algorithms and procedures (e.g., human activities) which are part of the overall process. *Instantiations* (i.e., physical assets) are the realizations of artifacts within their respective environments. They constitute the most concrete entities among the four different artifact types and are most suitable for empirical analysis including performance measures in terms of effectiveness and efficiency of the smart contract engineering process. In Table 4 we map the artifacts from Hevner et al. (2004) to the domain of smart contract engineering, with the respective artifacts shown in the left column and their manifestations within the domain in the right column.

5.2 Expert evaluation

In order to create an integrated process model that accounts for the specificities of smart contracts, we thoroughly analyzed the previous experiences from our experts following the guidelines for qualitative research (Mayring 2014) and design science research (Hevner et al. 2004). All but one interviewee (#7) were familiar with existing software engineering process models and confirmed that they actually build their

Table 2 Survey sample

ID	Role	Smart contracts experience	Software engineering experience	Highest degree	Blockchain type/domain
1	Independent developer	6 months	> 5 years	PhD (CS)	Ethereum with a focus on projects related to energy
2	Independent developer	More than 12 months	> 10 years	MSc (CS)	Ethereum with a focus on projects related to gambling
3	Head of the development team	6 months	> 10 years	MBA	Ethereum, hyperledger
4	Senior developer	6 months	> 10 years	PhD (CS)	Ethereum with a focus on prediction markets
5	Head of the development team	6 months	> 5 years	None	Ethereum, IOTA
6	Senior developer	More than 12 months	> 5 years	PhD (physics)	Ethereum with a focus on projects related to gambling
7	Junior developer	4 months	3 years	BSc (CS)	Ethereum with a focus on projects related to finance
8	Senior developer	6 months	> 5 years	None	Ethereum, did not want to disclose
9	Independent developer	8 months	> 5 years	None	Ethereum, hyperledger
10	Independent developer	4 months	> 20 years	PhD (CS)	Ethereum with a focus on projects related to gambling
11	Head of the development team	12 months	> 5 years	None	Ethereum, bitcoin

Table 3 Expert interviews

Professional background
What is your formal education?
What is your experience as a software developer/engineer?
Smart contracts development in general
Briefly describe the kind of smart contracts you develop.
Do you develop smart contracts for public/private or permissioned/permissionless blockchains?
For which blockchains do you develop smart contracts? Ethereum, Neo, Hyperledger, etc.?
Smart contract development process
Please describe how the roles of your smart contract development team are organized
How many team members are involved in the development of smart contracts?
Do you use a modeling approach?
Do you have a structured software development process? Which practices do you adopt?
What type of notation does your team use to document requirements for smart contracts?
Which programming languages and environments do you use when developing smart contracts?
Do you use tools to support the SW engineering? If yes, which one(s)?
Testing in smart contract development processes
Do you think that vulnerability to security incidents (e.g., due to software bugs) is a problem in current smart contract development?
Which parts are the hardest to test?
How often are software testing practices carried out during smart contract development? How often do you conduct the different types of test activities?
Do you automate your testing activities? To what extent? How do you incorporate security testing in this process?

own processes on them according to their needs—including modifications that are needed to account for the specifics of TADDL environments. All interviewees concurred that developing smart contracts is different from developing software in traditional ways and more comparable to developing hardware: "… it is much more like developing hardware that is shipped off to customers—without any chance to fix bugs once it has been sent off". Additionally, we found almost all described projects to be a mix of traditional software engineering and TADDL-specific engineering (cf "non-SC specific development" in Fig. 2). Additionally, all interviewees agreed that the submission of the smart contract to the live TADDL constitutes the most criticial part of the whole development process. For example, one interviewee stated that "… we even follow a paper-based approach where the entire team has to sign off before it is submitted. The entire team has to be present—it is very ceremonial".

Some interviewees had already perceived problems with existing testing approaches in a blockchain-based environment: "It is not possible to test under real-world conditions—we have a Testnet, but can never be sure to have similar conditions as in the real [TADDL] network". There was a general agreement regarding the importance of the analysis, specification, and validation of the implementation against the requirements: "We have tight feedback loops, where both the backend developers as well as the smart contract developers and our customers discuss the requirements and the implementation". Approval of a smart contract is in general

accompanied by extensive documentation: "… we document the entire [approval]. We print out all the relevant documentation and put it into a binder. We have test reports, coverage reports and, most importantly, the signed approval from our clients in there". Another interview partner reported challenges while launching their platform. As demand for the system exceeded expectations, too many server-side transactions were issued in a short period. As transaction costs were not adjusted, *thousands of transactions were* computed in the wrong order and thus failed. The engineers concluded that the calculation of transactions costs needs access to real-time data. Additionally, a queue for to-be-issued transactions was implemented within the platform.

Several major challenges were explicitly mentioned with regards to the availability of testing data from oracles. As some of our interviewees develop smart contracts that are intended (at least in theory) to run forever, they support the idea of a dedicated finalization phase—but were not able to specifically link it to their respective use cases. A comprehensive documentation is therefore important during contract runtime: "We definitely store everything, even after the smart contract is no longer under active development—especially since we do not know when something bad [referring to the DAO hack] might happen". Finally, based on the fact that smart contracts cannot be changed after deployment, a careful monitoring during runtime turns out to be crucial: "We have a dedicated watchdog [i.e., custom piece of software] that tracks the smart contract's spending and alerts us in case something odd happens."

Table 4 Mapping of Artifacts

Artifact	Manifestation within the domain
Construct	Trustless, append-only, decentralized, digital ledgers (TADDL) Cryptocurrency assets (i.e., tokens) Smart contract execution engine Smart contract expression language Actors (e.g., legal party, smart contract engineer, oracle, miner, legal expert) Wallets
Model	Smart contract code, templates, and patterns Transaction schemes The digital representation of assets Consensus and reward algorithms Interactions via transactions, function calls, oracle inputs
Method	Smart contract engineering (sub-)activities Iterations of the engineering process Simulation activities Test methods for smart contracts
Instantiation	An instance of the smart contract engineering process with its activities Operationalized smart contracts [e.g., instances of high-level languages compiled to Ethereum Virtual Machine bytecode (Wood 2014)] Results from smart contract test scenarios (e.g., reports and log files) Results from smart contract executions and simulations (e.g., transactions)

5.3 Roles, activities, and artifacts

To formally and constructively describe an engineering process, the Rational Unified Process (RUP) Framework (Kruchten 2004) can be used as a baseline for adaption since it is document-centric and reflects the smart contract development process. More specifically, the RUP is used to differentiate between three distinctive elements: First, *roles* pertain to individuals or groups performing activities within the process. Such roles, which might include smart contract engineers, software engineers, and legal experts, are responsible for the artifacts that are the outcome of their activities. Second, *activities* summarize a unit of work that must be performed. The outcome is the creation or update of artifacts. Third, *artifacts* denote the input and output of activities. Artifacts are created, modified, and used by the roles during the procedure and are either the final product, parts of it, or intermediate results. Examples of artifacts include concepts, models, source code, smart contract code, or documents such as performance reports.

Figure 1 illustrates the lifecycle of a smart contract. Sillaber and Waltl (2017) have shown that smart contract lifecycles start with *an implementation phase*, during which requirements are transformed into an implementation (Create and Adapt), verified against the requirements, and either approved for release or modified again. Once the smart contract is approved, it is published on the TADDL in the *submission stage*. In this phase, the smart contract is submitted and distributed within the TADDL network. From that point on, every entity with access to the TADDL can retrieve the contract and share it with other nodes. Once the smart contract has been spread throughout the network and is accepted by general consensus (i.e., it persists on the network), reverting or changing it requires—under ideal circumstances—substantial effort. The contract is now ready to be executed. In a paper contract analogy, this would be the signing of the contract, which is one important step to make the contract valid and enforceable. The *execution stage* of smart contracts is performed by miners or other participants of the TADDL, since the smart contract code is now accessible for all participants in the form of bytecode. To execute it, the smart contract is retrieved from the TADDL and carried out by the respective node (*compute*). Based on a given input, the output (e.g., a return value, a state transition, or a set of transactions) of a smart contract is computed, which is then stored and distributed within the network. This is similar to the "closing" of an offline contract. A smart contract can be executed as long as it is active. Its execution is resource-consuming and the nodes contributing computational power for its execution are rewarded according to the distributed ledger's reward scheme. In the *finalization stage,* the smart contract expires. This can happen either because the parties actively declare the smart contract as invalid (e.g., by withdrawing remaining funds or executing an appropriate function) or because of intrinsic conditions that make further executions impossible (e.g., time expiration; inability to pay the fees required for further execution). In this case, the smart contract remains in the TADDL, but can no longer be executed by the nodes. This means that the smart contract is disabled through a conditional exit that prevents future execution. This is akin to the "final" state of a business process, where specific properties (e.g. successful execution, final state) depend on the specific context.

5.4 An integrated smart contract engineering process

Figure 2 combines and summarizes our findings from the literature as well as the expert interviews into a comprehensive framework for the integrated smart contract engineering process. In the *conceptualization phase*, the preliminary scope and the goals of the smart contract are defined. The scope informs all involved parties about what will be, and what will not be, part of the smart contract and can be directly derived from traditional contractual requirements. This process is called requirements elicitation. The problem definition should also state the desired economic outcome(s). After reaching an agreement about the scope, the next step is the conceptual modeling: the transformation of the requirements of all involved parties into a smart contract model. In this phase, the conceptual model is created. The conceptual model defines classes of objects (e.g., wallets) and the desired relations between these objects and outcomes (e.g., transactions). The construction of the conceptual model will most likely uncover incomplete and contradictory aspects of the problem definition. Additionally, the modeling process may raise new questions for the involved parties to answer and resolve through negotiation. In either case, the problem definition should be adjusted.

After the conceptual modeling phase, the *implementation phase* starts. Here, the conceptual model is mapped onto an executable model (e.g., in Ethereum by using Solidity code) as existing smart contract patterns are identified, adapted and combined. It is critical to note that for performance and cost reasons, most business logic to be implemented will be executed outside the smart contract, within a "traditional software" application (termed "non-smart contract code"). The identification of those parts that should be included in the smart contract, and those that should be excluded, requires a thorough analysis of functional requirements as well as non-functional requirements for confidentiality, integrity, availability, scalability or efficiency. This analysis may even be performed as a formal risk analysis. In a banking application, for instance, core-banking functionality may be implemented as a smart contract, whereas data visualization on a traditional software stack.

An executable smart contract is not necessarily immediately correct and has to be reviewed, tested, and verified. Verification and simulation of the smart contract against the scope and stakeholder requirements are necessary to check whether the code contains errors, including programming errors and mal-adjusted parameters. For verification purposes ("Simulation, testing, code review"), various scenario-based executions can be simulated step-by-step in a private blockchain. Apart from verification, validation of the smart contract is also required. During validation, the simulation results of the smart contracts are compared to real-world contract states and stakeholder requirements. New insights may even lead to an adjustment of the problem definition and/or the conceptual model of the smart contract. A simulated

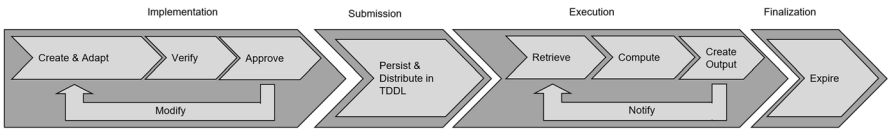

Fig. 1 Simplified lifecycle of a smart contract

smart contract found to be correct after validation is called a validated smart contract after the last round of consolidation.

Starting from the consolidated and validated smart contract, an instance of the smart contract can be frozen and submitted for execution in the live TADDL environment. Finally, in the *approval and execution phases*, the published smart contract is approved and executed in the TADDL and has to be monitored during runtime. In case the smart contract's behavior deviates from the stakeholders' requirements, appropriate change management mechanisms have to be activated: in extreme cases, the deactivation of the smart contract by depleting its funds and the creation of a new smart contract which better meets the stakeholders' requirements. Modifications of non-smart contract components are possible throughout the entire lifecycle of the smart contract. Although the smart contract becomes immutable after it has been submitted to the TADDL environment, the environment itself often provides opportunities to influence the outcome of smart contracts: for example, by influencing the call graph through a function registry or call delegation. The smart contract's runtime behavior is constantly monitored and managed in a change management process. Once the smart contract has reached the end of its life (e.g., by executing the "self-destruction" operation in the Ethereum blockchain), proper finalization can be confirmed in the *finalization phase* by validating whether the desired outcomes have been reached. Figure 2 further shows that feedback between phases is possible and frequently necessary. In practice, many phases will overlap. More specifically, specification, implementation, validation, and verification will go hand in hand once the appropriate tools are available to smart contract developers.

6 Discussion and implications

Smart contracts may well become the backbone of businesses based on blockchain and related technologies (Bailis 2017; Werbach and Cornell 2017). However, prior to the creation of industry-specific solutions, it is prudent to consider the general

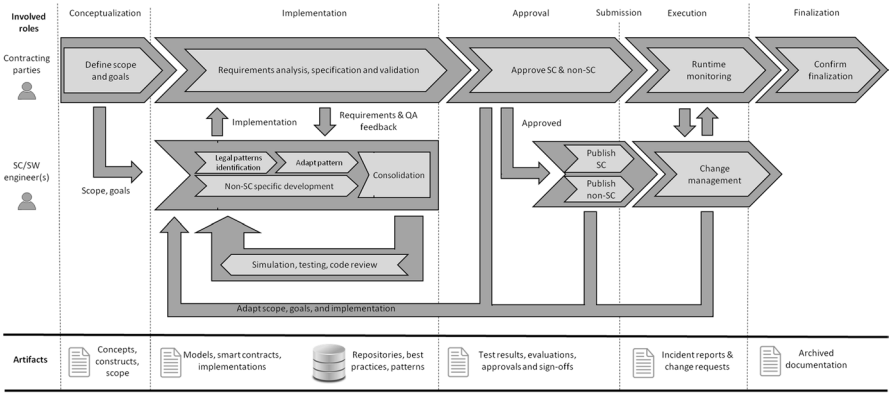

Fig. 2 Integrated smart contracts engineering process

characteristics of smart contracts and the roles they play during their lifecycle. In this paper, we therefore present an integrated process model for smart contracts that was developed and iteratively improved using findings from previous research and the feedback of eleven industry experts. This model highlights the important role of smart contracts during their lifecycle and thus supports quality management in software engineering. This is of utmost importance for the business community striving to use blockchain-based solutions, since immutable bugs in smart contracts with no possibility of rectification have been exploited in previous attacks. For example, the controversial hard-fork of the Ethereum blockchain, which basically nullified the effects of malicious transactions, poses an example of how laborious and far-reaching ex-post changes on blockchain can potentially be (Buterin 2016).

Our proposed smart contract engineering process is generic and is applicable to a wide variety of distributed ledger technologies. It is based on traditional software engineering process models and methodologies, such as the waterfall model as well as iterative models that have been successfully applied in a wide variety of use cases, and can be easily integrated with these existing models. For example, one cycle of the implementation phase can be aligned with a Scrum sprint (Schwaber and Beedle 2002). Traditional phase-oriented software engineering process models like the waterfall model typically progress linearly through an analysis, design, implementation, and testing phase. While the analysis, design and implementation phases align with our proposed conceptualization and implementation phases, special care has to be given to the testing phase of smart contracts, as this must be conducted and concluded prior to publishing the contract. Iterative software engineering process models typically iterate sequentially through the aforementioned four phases. The implementation phase proposed in this paper iterates through a pattern selection and adaption, development, consolidation, review, testing, and simulation phase, aligning these process activities with iterative software engineering process models.

Although further refinement of different aspects of the integrated process model may be necessary for specific applications, the integrated model as presented in this paper can immediately be applied in real-world industry settings. It can help smart contract engineers to better understand the strengths and weaknesses of their engineering processes and support them in further optimizing different process activities and artifacts such as software, models, and norms. Additionally, the process model can advance applied smart contract engineering processes and serve as a basis for critically investigating those processes in great detail, which is of practical value for any industry that needs to react fast while at the same time ensuring supreme software quality. Additionally, there are also implications for academic research. The engineering process model is accompanied by directions concerning the involved artifacts, roles, and interdependencies, and thereby lays the foundation for future research. This may include a detailed specification of the different roles and their required profiles.

7 Conclusions, limitations and future research

In this paper, we develop an integrative process model for smart contract engineering and describe its activities, roles, and artifacts. This model lays the foundation for further smart contract development, which has the potential to revolutionize many different industries. We argue that conventional software engineering process models do not provide adequate support for the trustless, append-only, and decentralized environment in which smart contracts are executed. Traditional process models do not account for the immutability of smart contracts after they are submitted, because they assume a (mostly) frictionless transition between software releases and that modifications of existing software releases are possible. Our smart contract engineering process accounts for these peculiarities of blockchain-based software development and consists of five sequential phases: (1) conceptualization, (2) implementation, (3) approval, (4) execution, and (5) finalization. These phases are derived from the properties of the underlying blockchain ecosystem. We propose new directions for smart contract engineering that focus on collaboration among domain experts, testing activities, quality assurance, and specialized workflow tools.

Currently, we see two major limitations of this research that deserve further attention. First, there are no validated measurements for the concepts of the process activities. An attempt was made to use existing process model artifacts and to make as few changes as possible. However, the construct validity of these artifacts cannot currently be guaranteed. Second, due to a lack of established best practices in smart contract engineering, an empirical evaluation of the hypothesized artifacts is not currently feasible.

Future research, therefore, needs to investigate if and how the proposed engineering process model can be tailored to and with different software engineering methodologies (e.g., Scrum, V-model). In this context, research could investigate how the development of smart code can be integrated with the development of traditional software code, as well as how risk analysis can support this integration. Furthermore, it is necessary to integrate this framework with existing work on testing and quality assurance in software engineering. An important aspect here is especially the role of simulation for quality assurance. The behavioral aspects of smart contract engineering have not yet received enough attention. While data is sparse, we have seen DevOps and "full-stack" software engineering behavior with many interviewees and many interesting patterns (e.g., randomness patterns or oracle patterns) that have been adapted from these and related disciplines that warrant future research. Furthermore, there is a pending need to cover the increasing demand for inter- TADDL transactions and developing secure applications that rely on more than one TADDL. Combining various approaches will lead to new insights into how best to cope with the challenges of modern blockchain-based software development and how smart contracts can be used to create viable business models.

Acknowledgements Open access funding provided by University of Innsbruck and Medical University of Innsbruck.

References

Anderson L, Holz R, Ponomarev A, Rimba P, Weber I (2016) New kids on the block: an analysis of modern blockchains. CoRR. arXiv preprint http://arxiv.org/abs/1606.06530

Atzei N, Bartoletti M, Cimoli T (2016) A survey of attacks on Ethereum smart contracts. IACR Cryptology ePrint archive 2016:1007

Atzei N, Bartoletti M, Cimoli T (2017) A survey of attacks on Ethereum smart contracts SoK. In: Proceedings of the international conference on principles of security and trust, Uppsala, Sweden, pp 164–186

Bailis P (2017) Research for practice: cryptocurrencies, blockchains, and smart contracts; hardware for deep learning. Commun ACM 60(5):48–51

Bartoletti M, Pompianu L (2017) An empirical analysis of smart contracts: platforms, applications, and design patterns. In: Proceedings of the international conference on financial cryptography and data security. Springer, Cham, pp 494–509

Baskerville R, Baiyere A, Gregor S, Hevner A, Rossi M (2018) Design science research contributions: finding a balance between artifact and theory. J Assoc Inf Syst 19(5):358–376

Beck R, Müller-Bloch C (2017) Blockchain as radical innovation: a framework for engaging with distributed ledgers as incumbent organization. In: Proceedings of the 50th Hawaii international conference on system sciences, Hawaii, HI, pp 5390–5399

Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R, Kern J, Marick B, Martin RC, Mellor S, Schwaber K, Sutherland J, Thomas D (2001) Manifesto for agile software development. http://agilemanifesto.org/. Accessed 20 Apr 2018

Bennett KH, Rajlich VT (2000) Software maintenance and evolution: a roadmap. In: Proceedings of the conference on the future of software engineering, pp 73–87

Bhargavan K, Delignat-Lavaud A, Fournet C, Gollamudi A, Gonthier G, Kobeissi N, Kulatova N, Rastogi A, Sibut-Pinote T, Swamy N, Zanella-Béguelin S (2016) Formal verification of smart contracts: short paper. In: Proceedings of the 2016 ACM workshop on programming languages and analysis for security, pp 91–96

Boehm B, Turner R (2005) Management challenges to implementing agile processes in traditional development organizations. IEEE Softw 22(5):30–39

Böhme R, Christin N, Edelman B, Moore T (2015) Bitcoin: economics, technology, and governance. J Econ Perspect 29(2):213–238

Bourque P, Fairley RE (2014) Guide to the software engineering body of knowledge (swebok (r)): version 3.0. IEEE Computer Society Press, Washington, DC

Braude EJ, Bernstein ME (2016) Software engineering: modern approaches. Waveland Press, Long Grove

Buterin V (2014) A next-generation smart contract and decentralized application platform. White paper. https://github.com/ethereum/wiki/wiki/White-Paper#decentralized-autonomous-organizations. Accessed 10 Jan 2018

Buterin V (2016) Hard fork completed. Ethereum Blog. https://blog.ethereum.org/2016/07/20/hard-fork-completed/. Accessed 17 Dec 2017

Chau PYK, Tam KY (1997) Factors affecting the adoption of open systems: an exploratory study. MIS Q 21(1):1–24

Clack CD, Bakshi VA, Braine L (2016) Smart contract templates: essential requirements and design options. CoRR. arXiv preprint http://arxiv.org/abs/1612.04496

Coinmarketcap (2018) Top 100 cryptocurrencies by market capitalization. https://coinmarketcap.com/. Accessed 17 Apr 2018

Delmolino K, Arnett M, Kosba A, Miller A, Shi E (2016) Step by step towards creating a safe smart contract: lessons and insights from a cryptocurrency lab. In: Clark J, Meiklejohn S, Ryan PYA, Wallach D, Brenner M, Rohloff K (eds) Proceedings of the international conference on financial cryptography and data security, pp 79–94

Deshpande A, Stewart K, Lepetit L, Gunashekar S (2017) Distributed Ledger technologies/blockchain: challenges, opportunities and the prospects for standards. Overview report. The British Standards Institution (BSI). https://www.bsigroup.com/PageFiles/508003/BSI_Blockchain_DLT_Web.pdf. Accessed 1 May 2018

Destefanis G, Marchesi M, Ortu M, Tonelli R, Bracciali A, Hierons R (2018) Smart contracts vulnerabilities: a call for blockchain software engineering? In International workshop on blockchain oriented software engineering (IWBOSE), Campobasso, Italy, pp 19–25

Edan Y, Pliskin N (2001) Transfer of software engineering tools from information systems to production systems. Comput Ind Eng 39(1–2):19–34

Egelund-Müller B, Elsman M, Henglein F, Ross O (2017) Automated execution of financial contracts on blockchains. Bus Inf Syst Eng 59(6):457–467

Fairfield JA (2014) Smart contracts, bitcoin bots, and consumer protection. Wash Lee Law Rev Online 71(2):35–50

Fanning K, Centers DP (2016) Blockchain and its coming impact on financial services. J Corp Account Finance 27(5):53–57

Frantz CK, Nowostawski M (2016) From institutions to code: towards automated generation of smart contracts. In: Proceedings of the IEEE international workshops on foundations and applications of self* systems, pp 210–215

Friedlmaier M, Tumasjan A, Welpe IM (2016) Disrupting industries with blockchain: the industry, venture capital funding, and regional distribution of blockchain ventures. In: Proceedings of the 51st Hawaii international conference on system sciences, Waikoloa, HI, pp 3517–3526

Fuggetta A, Di Nitto E (2014) Software process. In: Proceedings of the conference on future of software engineering, Hyderabad, India, pp 1–12

Hevner AR, March ST, Park J, Ram S (2004) Design science in information systems research. MIS Q 28(1):75–105

Idelberger F, Governatori G, Riveret R, Sartor G (2016) Evaluation of logic-based smart contracts for blockchain systems. In: Alferes J, Bertossi L, Governatori G, Fodor P, Roman D (eds) Rule technologies. Research, tools, and applications. RuleML 2016. Lecture notes in computer science, vol 9718. Springer, Cham, pp 167–183

IEEE (1995) 1074-1995—IEEE standard for developing software life cycle processes. IEEE. https://ieeexplore.ieee.org/document/490501/. Accessed 20 Mar 2018

Jacobson I, Booch G, Rumbaugh J, Rumbaugh J, Booch G (1999) The unified software development process, vol 1. Addison-Wesley, Reading

Johansen S (2018) A comprehensive literature review on the Blockchain as a technological enabler for innovation. Working paper, Mannheim University

KingOfTheEther (2016) Post-mortem investigation. https://www.kingoftheether.com/postmortem.html. Accessed 10 Apr 2018

Kiviat TI (2015) Beyond bitcoin: issues in regulating blockchain transactions. Duke Law J 65:569–608

Klöhn L, Parhofer N, Resas D (2018) Initial coin offerings (ICOs). Z Bankr Bankwirtsch 30(2):89–106

Koulu R (2016) Blockchains and online dispute resolution: smart contracts as an alternative to enforcement. SCRIPTed 13(1):40–69

Kruchten P (2004) The rational unified process: an introduction. Addison-Wesley Professional, Boston

Kuhrmann M, Diebold P, Münch J, Tell P, Garousi V, Felderer M, Trektere K, McCaffery F, Linssen O, Hanser E, Prause CR (2017) Hybrid software and system development in practice: waterfall, scrum, and beyond. In: Bendraou R, Raffo D, LiGuo H, Maggi FM (eds) Proceedings of the 2017 international conference on software and system process, Paris, France, pp 30–39

Lacity MC, Janson MA (1994) Understanding qualitative data: a framework for text analysis methods. J Manag Inf Syst 11(2):137–155

Lee G, Xia W (2010) Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility. MIS Q 34(1):87–114

Leitner A, Ciupa I, Oriol M, Meyer B, Fiva A (2007) Contract driven development = test driven development—writing test cases. In: Crnkovic I, Bertolino A (eds) Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering (ESEC-FSE '07). New York, NY, USA, pp 425–434

Luu L, Chu D-H, Olickel H, Saxena P, Hobor A (2016) Making smart contracts smarter. In: Weippl E, Katzenbeisser S, Kruegel C, Myers A, Halevi S (eds) Proceedings of the 2016 ACM Sigsac conference on computer and communications security, pp 254–269

March ST, Smith GF (1995) Design and natural science research on information technology. Decis Support Syst 15(4):251–266
Marjanovic O, Milosevic Z (2001) Towards formal modeling of e-contracts. In: Proceedings of the fifth IEEE international conference on enterprise distributed object computing, pp 59–68
Mayring P (2014) Qualitative content analysis: theoretical foundation, basic procedures and software solution. Dissertation. http://nbn-resolving.de/urn:nbn:de:0168-ssoar-395173. Accessed 1 May 2018
Mens T, Guehéneuc Y-G, Fernández-Ramil J, D'Hondt M (2010) Guest editors' introduction: software evolution. IEEE Softw 27(4):22–25
Moyano JP, Ross O (2017) KYC optimization using distributed ledger technology. Bus Inf Syst Eng 59(6):411–423
Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. https://bitcoin.org/en/bitcoin-paper. Accessed 12 Aug 2017
Narayanan A, Clark J (2017) Bitcoin's academic pedigree: the concept of cryptocurrencies is built from forgotten ideas in research literature. ACM Queue 15(4):1–30
Notheisen B, Cholewa JB, Shanmugam AP (2017) Trading real-world assets on blockchain. Bus Inf Syst Eng 59(6):425–440
Pesch PJ, Sillaber C (2017) Distributed Ledger, Joint Control? – Blockchains and the GDPR's Transparency Requirements. Comput Law Rev Int 18(6):166–172. https://doi.org/10.9785/cri-2017-0602t
Porru S, Pinna A, Marchesi M, Tonelli R (2017) Blockchain-oriented software engineering: challenges and new directions. In: Uchitel S, Orso A, Robillard M (eds) Proceedings of the 39th international conference on software engineering companion, Buenos Aires, Argentina, pp 169–171
Prusty SK, Mohapatra PKJ, Mukherjee CK (2017) House of strategy: a model for designing strategies using stakeholders' opinion. Comput Ind Eng 108:39–56
Rajlich VT, Bennett KH (2000) A staged model for the software life cycle. Computer 33(7):66–71
Rückeshäuser N (2017) Do we really want blockchain-based accounting? Decentralized consensus as enabler of management override of internal controls. In: Leimeister JM, Brenner W (eds) Proceedings der 13. Internationalen Tagung Wirtschaftsinformatik (WI 2017), St. Gallen, Switzerland, pp 16–30
Schwaber K, Beedle M (2002) Agile software development with scrum. Prentice Hall, Upper Saddle River
Seijas PL, Thompson SJ, McAdams D (2016) Scripting smart contracts for distributed ledger technology. https://eprint.iacr.org/2016/1156.pdf. Accessed 1 May 2018
Siegel D (2016) Understanding the DAO attack. https://www.coindesk.com/understanding-dao-hack-journalists/. Accessed 10 Apr 2018
Sillaber C, Waltl B (2017) The life cycle of smart contracts in blockchain ecosystems. Datenschutz Datensicherheit DuD 41(8):497–500
Stark J (2015) Product lifecycle management. Springer, London
Swan M (2015) Blockchain: blueprint for a new economy. O'Reilly Media, Sebastopol
Szabo N (1997) The idea of smart contracts. Nick Szabo's papers and concise tutorials. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html. Accessed 1 May 2018
Tapscott D, Tapscott A (2016) Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world. Penguin, New York
Treiblmaier H, Zeinzinger Z (2018) Understanding the blockchain through a gamified experience: a case study from Austria. In: 25th European conference on information systems, June 23–28, Portsmouth: UK
Tschorsch F, Scheuermann B (2016) Bitcoin and beyond: a technical survey on decentralized digital currencies. IEEE Commun Surv Tutor (COMST) 18(3):2084–2123
Vidgen R, Wang X (2009) Coevolving systems and the organization of agile software development. Inf Syst Res 20(3):355–376
Wang H, Chen K, Xu D (2016) A maturity model for blockchain adoption. Financ Innov 2(12):1–5
Werbach K, Cornell N (2017) Contracts ex machina. Duke Law J 67(2):313–382
Wood G (2014) Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, pp 1–32
Xu X, Weber I, Staples M, Zhu L, Bosch J, Bass L, Pautasso C, Rimba P (2017) A taxonomy of blockchain-based systems for architecture design. In: Proceedings of the IEEE international conference on software architecture, Gothenburg, Sweden, pp 243–252
Zakarian A, Kusiak A (2001) Process analysis and reengineering. Comput Ind Eng 41(2):135–150

Affiliations

Christian Sillaber[1] · Bernhard Waltl[2] · Horst Treiblmaier[3] · Ulrich Gallersdörfer[2] · Michael Felderer[1,4]

Horst Treiblmaier
horst.treiblmaier@modul.ac.at

[1] University of Innsbruck, Innsbruck, Austria

[2] TU Munich, Munich, Germany

[3] MODUL University Vienna, Am Kahlenberg 1, 1190 Vienna, Austria

[4] Blekinge Institute of Technology, Karlskrona, Sweden