Software Engineering Research for Blockchain-Based Systems

Dr Mark Staples
Research Group Leader, Software Systems
Conjoint Associate Professor, School of Computer Science & Engineering, UNSW

www.csiro.au
Data61 – Australia Digital Innovation

1100+ people
415+ research students
38 university partners
31 government partners
91 corporate partners

- Data capture
- Communications & networks
- Hardware & software
- Cybersecurity
- Data, statistics, ML
- Decision sciences, AI
- Behavioral economics & cogsci
50 Years of Software Engineering
“The phrase ‘software engineering’ was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering.”

“...discussions [...] under the three main headings: Design of Software, Production of Software, and Service of Software.”
Design, Production, Service

• Design Criteria
  – Use requirements
  – Reliability
  – Logical completeness

• Design Strategies
  – Sequencing the process
  – Structuring the design
  – Feedback with monitoring & simulation
  – High-level languages

• Communication & Management
Design, Production, Service

• Problems of scale
• Problems of reliability
• Management
  – Production planning
  – Personnel factors
  – Production control
  – Internal communication
• Technical aspects
  – Tools
  – Concepts
  – Performance monitoring

Figure 1. From Nash: Some problems in the production of large-scale software systems.
Design, Production, Service

• Service
  – Service goals
  – Initial release
  – Frequency of release

• Replication, Distribution, Maintenance

• Evaluation
  – Acceptance testing
  – Performance monitoring
  – Feedback from users
  – User documentation
  – Reprogramming
Special Topics, Working Papers

- Software crisis
- Education
- Software pricing
- Mass-produced components
- Checklist for planning software system production
- Complexity controlled by hierarchical ordering of function and variability
- Thoughts on the sequence of writing software
- The testing of computer software
- Performance monitoring and systems evaluation
- Towards a methodology of computing systems design
Software System Fashions

Google Scholar publication counts for topics co-occurring with “software engineering”. For 4-year periods since 1989, normalised relative to counts for “software engineering” in each period (2013-106 baseline). For 2017 period, projected from H1.
Blockchain
First, What Is A Blockchain?

(demo)
What Does a Blockchain Do?

• Functionally, blockchains are...
  
• A database (ledger)
  – Record of transactions

• A compute platform
  – “Smart contracts”

• Distributed, and no central owner
What’s New About Blockchain?

• What does blockchain bring that is new, and makes a difference to software engineering research?

• What are the new software-engineering research challenges for blockchain-based systems?
  – Some threads of research at Data61
    • Treasury projects
    • Architecting application on blockchain
    • Using smart contracts for business process execution
    • Formal specification and verification of smart contracts
  – Some other research areas
Today’s Focus: Not Computer Science

• Not looking at topics on new/ improved/ analyses of:
  – Consensus algorithms
  – Hash functions
  – Crypto schemes
  – Zero-knowledge schemes
  – P2P networking
  – Smart contract virtual machines
Treasury projects


“Distributed Ledgers: Scenarios for the Australian Economy Over the Coming Decades”, Hanson, R. T., Reeson, A., Staples, M., 2017

www.data61.csiro.au/blockchain
Technical Perspectives

Software Architecture

Non-Functional Properties
- Security, Performance, …

Dependable Software Systems

Trusted and Trustworthy Systems
- Risk, Evidence, Assurance, …

Three Illustrative Use Cases

Supply Chain
Registries
Remittances
Non-Functional Properties

There are two kinds of requirements:

1. **Functional Requirements** (i.e. inputs vs. outputs)
2. **Non-Functional Requirements** (a.k.a. “Qualities”, or “-ilities”)
   - e.g. Performance
     - latency, throughput, scalability (of latency, throughput, ...), timeliness, ...
   - e.g. Security
     - confidentiality, integrity, availability, non-repudiation, privacy, ...
   - e.g. Usability, Reliability, Modifiability, ...
Why Non-Functional Properties Matter

Regulatory Requirements  ↔  System Performance  ↔  Market Opportunities

Latency  Throughput
Cost  Throughput
Integrity  Confidentiality

New customers
New segments
New industries

System Performance

Market Opportunities

New customers
New segments
New industries
Software Architecture

Non-Functional Properties arise from Architectural Design Choices

Choose your design,
Choose your trade-offs
Blockchain vs Conventional Databases

• Logically centralised; Physically and organisationally decentralised

• Trade-offs for Various Non-Functional Properties
  (+) Integrity, Non-repudiation
  (-) Confidentiality, Privacy
  (-) Modifiability
  (-) Throughput/ Scalability/ Big Data
  (+ read/ - write) Availability/ Latency
Blockchains are Not Stand-Alone Systems

UI for humans

Key management

IoT integration

Blockchain is a component

Legacy systems

BIG DATA

Auxiliary databases

private data
We (Will) Rely on Blockchain-Based Systems

• The DAO failure hurt
  – Cost USD$60M? + fork

• Huge future economic value (or else there’s no point!)
  – e.g. supply chain, asset registries, settlement, ...

• Security-critical and Safety-critical use cases
  – e.g. e-health records, pharma supply chain, IoT management, ...
What is “Trust”? 

Dependable Software Systems says...

• **Trusted System**
  – A system you have chosen to rely on to fulfil a goal
    • When it fails, you suffer harm or loss

• **Trustworthy System**
  – A system where you have evidence it will not fail

“Trust” means accepting exposure to risk
Trustworthy Blockchain-Based Systems?

• What is good evidence that blockchain-based systems will do what we need?
  – Functional correctness
  – Non-Functional properties

• How do we get regulatory acceptance?
Assurance: Evidence & Acceptance

• Test blockchains in the rain

• Technologically-neutral regulation and policy

• But look carefully at blockchain-specific risks

• Need indicative guidance on regulatory acceptance of blockchain-based systems

• There are open questions about blockchain governance

• Increase R&D on trustworthy blockchains!
Potential Use Cases

• Financial Services
  – Digital currency
  – (International) payments
  – Reconciliation
  – Settlement
  – Markets
  – Trade finance

• Government Services
  – Registry & Identity
  – Grants & Social Security
  – Quota management
  – Taxation

• Enterprise and Industry
  – Supply chain
  – IoT
  – Metered access
  – Digital rights 7 IP
  – Data management
  – Attestation
  – Inter-divisional accounting
  – Corporate Affairs

• Three Illustrative Cases Selected
  1. Agricultural supply chain
  2. Open data registry
  3. Remittance payments
Remittances

- Blockchains may help reduce cost and time of remittances, but challenges remain for solutions to KYC

- Blockchains and smart contracts may make it possible to create ‘programmable money’
Supply Chain

• Supply chains are a highly promising domain for blockchains

• May enable significant opportunities for trade finance and insurance
Registries

- Blockchains can help to federate registries

- Sometimes too much integrity causes problems
Other Findings

• Blockchains have a different cost model

• Private blockchains are often not private enough

• Public blockchains might be OK for some purposes, even in regulated industries

• Blockchains have limitations – sometimes that doesn’t matter!
# Busting Blockchain Myths

<table>
<thead>
<tr>
<th>Myth</th>
<th>Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solves Every Problem</td>
<td>A kind of database</td>
</tr>
<tr>
<td>Trustless</td>
<td>Can shift trust and spread trust</td>
</tr>
<tr>
<td>Secure</td>
<td>Focus is Integrity, not Confidentiality</td>
</tr>
<tr>
<td>Smart contracts are legal contracts</td>
<td>May help execute parts of some legal contracts</td>
</tr>
<tr>
<td>Immutable</td>
<td>Many only offer probabilistic immutability</td>
</tr>
<tr>
<td>Need to waste electricity</td>
<td>Emerging blockchains are more efficient</td>
</tr>
<tr>
<td>Are inherently unscalable</td>
<td>Emerging blockchains are more scalable</td>
</tr>
<tr>
<td>If beneficial, will be adopted</td>
<td>Adoption can be hampered by FUD</td>
</tr>
</tbody>
</table>
Architecting Applications on Blockchain

“Comparing blockchain and cloud services for business process execution”, P. Rimba, A. B. Tran, I. Weber et al., ICSA2017
When to Use Blockchain? Trade-offs?

• Architecting applications on Blockchain:
  – Taxonomy and design process [1]
  – “Cost of Distrust”: how much more expensive is blockchain? [2]
  – Latency: simulation under changes [3]

• Model-driven development of smart contracts
  – Business process execution [4,5]
  – Model-based generation of registries and UIs: “Regerator” [6]
Blockchain-related design decisions regarding (de)centralisation, with an indication of their relative impact on quality properties
Legend: ⊕: Less favourable, ⊕⊕: Neutral, ⊕⊕⊕: More favourable

<table>
<thead>
<tr>
<th>Design Decision</th>
<th>Option</th>
<th>Impact</th>
<th>#Failure points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Centralised</td>
<td>Services with a single provider (e.g., governments, courts)</td>
<td>⊕⊕⊕</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Services with alternative providers (e.g., banking, online payments,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cloud services)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partially Centralised &amp; Partially Decentralised</td>
<td>Permissioned blockchain with permissions for fine-grained operations</td>
<td>⊕⊕</td>
<td></td>
</tr>
<tr>
<td></td>
<td>on the transaction level (e.g., permission to create assets)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Permissioned blockchain with permissioned miners (write), but</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>permission-less normal nodes (read)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fully Decentralised</td>
<td>Permission-less blockchain</td>
<td>⊕⊕⊕</td>
<td>Majority</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(nodes, power,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stake)</td>
</tr>
<tr>
<td>Verifier</td>
<td>Single verifier trusted by the network (external verifier signs valid</td>
<td>⊕⊕</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>transactions; internal verifier uses previously-injected external state)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>M-of-N verifier trusted by the network</td>
<td>⊕⊕⊕</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>Ad hoc verifier trusted by the participants involved</td>
<td>⊕⊕</td>
<td>1 (per ad hoc</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>choice)</td>
</tr>
</tbody>
</table>
Blockchain-related design decisions regarding storage and computation, with an indication of their relative impact on quality properties

Legend: ⊕: Less favourable, ⊕⊕: Neutral, ⊕⊕⊕: More favourable

<table>
<thead>
<tr>
<th>Design Decision</th>
<th>Option</th>
<th>Fundamental properties</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cost efficiency</td>
</tr>
<tr>
<td>Item data</td>
<td>On-chain</td>
<td>⊕⊕⊕⊕</td>
<td>⊕</td>
</tr>
<tr>
<td></td>
<td>Embedded in transaction (Bitcoin)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Embedded in transaction (Public Ethereum)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Smart contract variable (Public Ethereum)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Smart contract log event (Public Ethereum)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Off-chain</td>
<td>⊕</td>
<td>~KB Negligible</td>
</tr>
<tr>
<td></td>
<td>Private / Third party cloud</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Peer-to-Peer system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item collection</td>
<td>On-chain</td>
<td>⊕⊕⊕⊕</td>
<td>⊕⊕⊕ (public)</td>
</tr>
<tr>
<td></td>
<td>Smart contract</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Separate chain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computation</td>
<td>On-chain</td>
<td>⊕⊕⊕⊕</td>
<td>⊕</td>
</tr>
<tr>
<td></td>
<td>Transaction constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Smart contract</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Off-chain</td>
<td>⊕</td>
<td>⊕⊕⊕⊕</td>
</tr>
<tr>
<td></td>
<td>Private / Third party cloud</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Proposed design process (using Taxonomy)

Has trusted authority?

- Yes
  - Can it be decentralised?
    - Yes
      - How to decentralise the authority?
        - (IV.A)
    - No
      - Use traditional database

- No
  - Storage and computation: on-chain vs. off-chain
    - (IV.B)

Blockchain configurations (V.C)

- Need a new blockchain?
- Need multiple blockchains?
- What type?
- What block size and frequency?
- What consensus protocol?
- What data structure?

Other design decisions (V.D)

- What incentive?
- Need anonymity mechanism?

Where to deploy? (IV.D)
Cost of Distrust

• RQ: How much more expensive is blockchain over Cloud services?
  – Lens: business process execution
  – AWS SWF vs. Ethereum public blockchain
    • In both cases: pay per instruction
  – Experiments on two use cases:
    – Incident management (literature)
      • 32 instances on public Ethereum vs. 1000 runs on SWF
    – Invoicing (industry, 5316 log traces, 65K events)
      • Full log replayed on SWF and private Ethereum

• Result:
  – 2 orders of magnitude more expensive to use blockchain
  – ~US$ 0.35 per process instance on public blockchain
    • outweighed by cost of escrow (if needed) for about US$ 10 of value
Latency simulation

• Goal: predict latency for blockchain-based applications

• Approach: Architecture performance modeling
  – Paladio Component Models with individual latency distributions + connections + probability of branching
  – Allows changing the models for What-If analysis
  – For instance: change inter-block time on private blockchain – what does that mean for overall application latency?
Latency simulation
Latency: what if we change required number of confirmation blocks?

1 block

6 blocks

12 blocks
Using Smart Contracts for Business Process Monitoring and Execution


Motivation

• Integration of business processes across organizations: a key driver of productivity gains

• Collaborative process execution
  – Doable when there is trust – supply chains can be tightly integrated
  – Problematic when involved organizations have a lack of trust in each other
    → if 3+ parties should collaborate, where to execute the process that ties them together?
  – Common situation in “coopetition”
Motivation: example

Issues:
- Knowing the status, tracking correct execution
- Handling payments
- Resolving conflicts

→ Trusted 3rd party?
→ Blockchain!
Approach in a nutshell

• Goal: execute collaborative business processes as smart contracts
  – Translate (enriched) BPMN to smart contract code
  – Triggers act as bridge between Enterprise world and blockchain
  – Smart contract does:
    • Independent, global process monitoring
    • Conformance checking: only expected messages are accepted, only from the respective role
    • Automatic payments & escrow
    • Data transformation
    • Encryption
Architecture
Runtime

• Instantiation:
  – New *instance contract* per process instance
  – Assign accounts to roles during initialization
  – Exchange keys and create secret key for the instance

• Messaging:
  – Instead of direct WS calls, send through triggers & smart contract
  – Instance contract handles:
    • Global monitoring
    • Conformance checking
    • Automated payments*
    • Data transformation*
Translator

• Translate subset of BPMN elements to *Solidity*
  – BPMN Choreography diagrams or regular BPMN models with pools
1. Translate BPMN control flow to Wfnet
2. Capture dataflow and conditions
3. Reduce WFnet and annotate dataflow
4. Translate into Solidity code
   - Status of the process is kept in a bit vector
   - Updates are bit-wise operations
Bit vector check: does pos 1 have value “1”?

```solidity
class BPMJContract {
    uint marking = 1;
    uint predicates = 0;

    function CheckApplication( input params ) returns (bool) {
        if (marking & 2 == 2) { // is there a token in place p1?
            // Task B's script goes here, e.g. copy value of input params to contract variables
            uint tmpPreds = 0;
            if ( -eval P - ) tmpPreds |= 1; // is loan application complete?
            if ( -eval Q - ) tmpPreds |= 2; // is the property pledged?
            step(
                marking & uint(~2) | 12, // New marking
                predicates & uint(~3) | tmpPreds // New evaluation for "predicates"
            );
            return true;
        }
        return false;
    }
}
```

Bit vector update: set pos 1 to “0”  
set pos 2 and 3 to “1”
Payments, escrow, data handling

• Payment / escrow using crypto-coins:
  – Instance contract has an account
  – Only the contract code governs what gets paid out, but anyone can pay in
  – Contract can base validity of transactions on associated payments
  – Anyone can see balance of the contract – enables trust:
    • All parties know when the money is there
    • The contract code specifies who gets paid how much, and when

• Data:
  – Status update data has to readable for the contract, most other data can be encrypted
    • Data used in conditions has to be readable
  – Sending data over blockchain is costly – can split on-chain vs. off-chain
    • On-chain: URI to the data + hash
    • Off-chain: reachable and addressable data store (IPFS, S3, …)
Evaluation

• 4 use case processes, 1 from industry with 5316 traces and 65K events

• Executed ~150K transactions on private and 256 TXs on public Ethereum blockchain

• Look at Correctness, Cost, Latency, Throughput
Evaluation – Cost

• Per process instance: cost on average of 0.0347 Ether (~$0.40) before optimization

• After optimization:
  – About 25% reduction for industrial process (Opt-CF)
  – Up to 75% reduction if smart contract can be reused (Opt-Full)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoicing</td>
<td>5316</td>
<td>Default</td>
<td>1,089,000</td>
<td>33,619</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opt-CF</td>
<td>807,123</td>
<td>26,093</td>
<td>-24.97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opt-Full</td>
<td>54,639</td>
<td>26,904</td>
<td>-75.46</td>
</tr>
<tr>
<td>Supply chain</td>
<td>62</td>
<td>Default</td>
<td>304,084</td>
<td>25,564</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opt-CF</td>
<td>298,564</td>
<td>24,744</td>
<td>-2.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opt-Full</td>
<td>54,248</td>
<td>25,409</td>
<td>-42.98</td>
</tr>
<tr>
<td>Incident mgmt.</td>
<td>124</td>
<td>Default</td>
<td>365,207</td>
<td>26,961</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opt-CF</td>
<td>345,743</td>
<td>24,153</td>
<td>-7.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opt-Full</td>
<td>54,499</td>
<td>25,711</td>
<td>-57.96</td>
</tr>
<tr>
<td>Insurance claim</td>
<td>279</td>
<td>Default</td>
<td>439,143</td>
<td>27,310</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opt-CF</td>
<td>391,510</td>
<td>25,453</td>
<td>-8.59</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opt-Full</td>
<td>54,395</td>
<td>26,169</td>
<td>-41.14</td>
</tr>
</tbody>
</table>
Evaluation – Latency

• Primary source of latency: mining time
  – Public Ethereum blockchain: median time between blocks set to 13-15s
  – Private blockchain: can control it
Evaluation – Throughput

Optimized implementation can fill up each block

- Limit: blockchain network-defined gas limit

- Unoptimized
- Control flow optimized
- Fully optimized

Number of transactions vs. Blocks since start of experiment
Formal Specification and Verification of Smart Contracts

Formal Specification & Verification

Program testing can be used to show the presence of bugs, but never to show their absence!

- Dijkstra, ’60s/’70s

• Formal methods is logical reasoning about all possible behaviours of software

• Is the strongest kind of evidence that software is correct

real use/needs → spec. → code → physical system → formal spec.

\[ \forall x. A(x) \rightarrow R(x) \]

\[ A(x), Q(x) = R(x) \]

\[ \forall x. A(x) \rightarrow Q(x) \]

formal verification proof code semantics
seL4 verified @ Data61

- **seL4**: embedded systems microkernel
  - high-performance, in C and ASM
  - capability-based security
  - 2009 formal proof of correctness
  - ongoing extensions and proofs

- Demonstrated in DARPA HACMS
  - unmanned helicopter
  - red team couldn’t hack

---

**Diagram:**
- Trusted
  - Crypto (native)
  - Comms (native)
  - Network Stack (native)
  - Mission Manager (native)

- Untrusted
  - Camera & WiFi (Linux VM)
  - VMM

---

**Logos:**
- Rockwell Collins
- galois
- DATA 61
- CSIRO
- University of Minnesota
- Boeing
Limits of Formal Methods

• sel4 is “bug free”*
  – No null pointer dereferences
  – No undefined behaviour
  – All behaviours conform to spec
  – Security policies are enforced
  – ...

(*) conditions apply:
  – Correct hardware
  – Correct boot code
  – DMA off or trusted
  – ...

\[ \forall x. A(x) \rightarrow Q(x) \]
\[ A(x), Q(x) = R(x) \]
\[ \forall x. A(x) \rightarrow R(x) \]

real use/needs

formal spec.

verification proof

code semantics

physical system

58 | Software Engineering Research for Blockchain-Based Systems | Data61, CSIRO
Formalising Smart Contracts

- real user needs & real legal contracts
- Specification Language
- Source Language
- Bytecode Language
- Virtual Machine Interpreter

Logic-based smart contracts
Idelberger et al., 2016
Al Khalil et al., 2017

EVM in HOL
Hirai, 2016, 2017

Idris type safety
Pettersson & Edström, 2016

Pact type safety
Popejoy, 2016

Solidity & EVM via F*
Bhargavan et al., 2016

EVM symbolic execution
Luu et al., 2016

Imandra & EVM
Ignatovich & Passmore, 2015

Determinism, Immutability,
Cross-checking & Social Consensus

mining network
Smart Contract Verification

• Data61 is now contributing to Yoichi Hirai’s formalization of EVM
  – Within the Data61 group that verified and maintains seL4
  – https://github.com/pirapira/eth-isabelle

• “Proof engineering”: cheaper & more scalable formal verification
  – Verified code frameworks for compositional code & compositional proof
  – Formal decompilation from bytecode to higher-level languages
  – Metric-driven proof improvement and estimation

• Linking to formal specification of smart contracts...
"Code is Law" Lessig  "The Law is an Ass" trad.
"Code is an Ass"
Smart Contract Specification & Generation

• Create and verify high-level models; transform to code & proofs
  – Domain-Specific Languages/ Model-Driven Architecture/ Formal Synthesis

• Process models, for inter-org. coordination

• Schemas, for Registries

• Declarative logics, for Legal contracts

Weber et al., 2016
García-Bañuelos et al., 2017
Tran et al., 2017
Idelberger et al., 2016
Trustworthy Blockchain-Based Systems?

• Smart Contracts are just part of the technology stack
• Functional correctness is just one aspect of fitness
• Need system-level assurance arguments about all qualities
Wrapping Up...
Other Research Areas? Porru et al. (2017)

• Blockchain challenges
  – New professional roles
  – Security and reliability
    • Smart contract testing
    • Blockchain transaction testing
  – Software architecture
  – Modeling languages
  – Metrics

• New research directions
  – Testing
  – Collaboration
  – Enhancement of testing & debugging
  – Creation of software tools for smart contract languages

NATO 1968 for Blockchain?

<table>
<thead>
<tr>
<th>Design</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Model-driven blockchain-based systems</td>
<td>• Release management</td>
</tr>
<tr>
<td>• Smart contract specification</td>
<td>• Blockchain monitoring</td>
</tr>
<tr>
<td>• Blockchain architecture &amp; design</td>
<td>• Inter-chain operations</td>
</tr>
<tr>
<td>• Cost, performance prediction</td>
<td>• Recovery from forks/uncles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Production</th>
<th>Other topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Testing for smart contracts &amp; txns</td>
<td>• Professional roles</td>
</tr>
<tr>
<td>• Estimation &amp; metrics for blockchain dev</td>
<td>• Education</td>
</tr>
<tr>
<td>• Smart contract tooling</td>
<td>• Handling blockchain software crises</td>
</tr>
<tr>
<td>• Performance and scaling</td>
<td></td>
</tr>
</tbody>
</table>
References


11. V. Gramoli, *On the Danger of Private Blockchains,* DCC12016