Trent McConaghy Follow
AI*blockchain. Founder @OceanProtocol | @BigchainDB www.trent.st
Mar 1 · 17 min read

Towards a Practice of Token Engineering

Methodology, Patterns & Tools. TE Series Part II.

1. Introduction

In my previous article, I described *why* we need to get incentives right when we build these tokenized ecosystems. Here, I ask: *how* do we design incentives for tokenized ecosystems? And actually since incentives are the heart of tokenized ecosystems, it's really: *how do we design tokenized ecosystems?* And, how do we *analyze* and *verify* them?

This article is a first stake in the ground towards a practice of **token engineering**: the theory, practice and tools to analyze, design, and verify tokenized ecosystems.

The first section of this article relates token designs to other fields and explains why "engineering". The rest of this article is an attempt to draw us closer to this goal, by leveraging existing fields in three main ways:

- We can frame **token design as optimization design**, then use optimization design methodology.
- Inspired by **software engineering patterns**, we can document emerging patterns for token design.
- **Simulators and CAD tools for circuit design** has helped engineers analyze, design, and verify wickedly complex chips. We can look forward to similar tools for tokenized ecosystems.

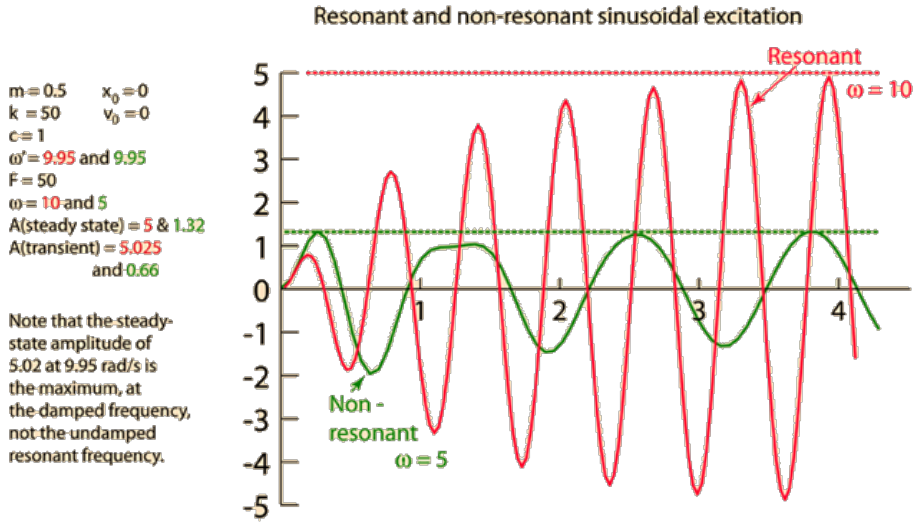
2. Engineering, Game Theory, and More

This section relates token design to other fields.

2.1 The Tacoma Narrows Bridge

In the first week of my engineering studies, our solemn-faced professor showed us this:

How did the bridge collapse, leading to the death of many? The designers *did* anticipate for wind, after all. However, they failed to anticipate that the particular wind patterns would set up *resonance* with the bridge itself. When you have a constant force applied to a system in resonance, the amplitude of the resonance *grows* over time. The figure below illustrates, where green = non-resonant and red = resonant = disaster.



[Image from here]

The video was to teach about *responsibility*. The designers of the bridge could have prevented the disaster by being *thorough,* and applying appropriate theory, practice, and tools. Other professors showed that video several times over the years during my studies. Those viewings culminated in a ceremony to receive iron rings upon graduation. All Canadian engineers have these rings. According to legend, the rings are forged from the metal of a collapsed bridge.

2.2 Game Theory and Mechanism Design

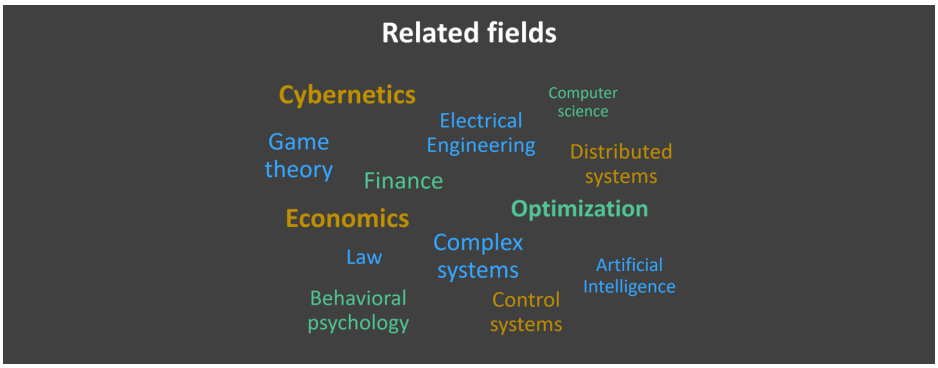
Game Theory is a scientific field that **analyzes** incentives, from an economic standpoint. It has a counterpart in economics for **designing (synthesizing)** incentivized systems, called **Mechanism Design**. In fact Mechanism Design is really *the* field for design of tokenized ecosystem, in theory. Researchers in that field have come up with a lot of great theory over the years, at literally Nobel prize levels of quality.

However, traditionally there hasn't been a good way to reconcile that theory with *practice*. After all, how often does an academic economist (or anyone, really) get a chance to deploy an economy? Yet this is the exact problem we are confronted with in designing tokenized ecosystems.

However, it turns out that if you zoom in on Mechanism Design with a few practical constraints, you end up with Optimization Design! People doing Optimization Design have a tremendous amount of practical experience deploying optimizer systems over the years. Myself included: both my first and second startups (ADA, Solido) did exactly that, for use in industrial-grade circuit design. I'll discuss optimization design in a later article (TE II).

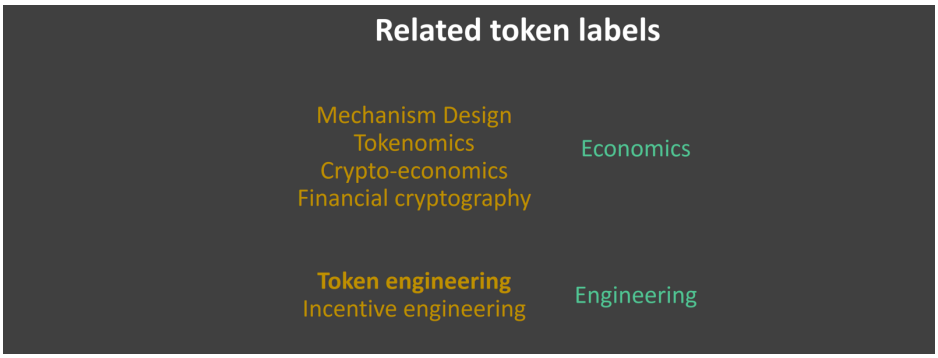
2.3 Other Related Fields

Many other fields have something to say about token design as well; at the very least in the sense that experts from those fields will find that many of their skills translate well to token design. These include everything from electrical engineering to complex systems, from economics to AI. I list a few below. And of course many of them have roots in good ol' cybernetics.



2.4 Engineering and Token Design

What should we be calling this field in which we design tokenized ecosystems? I list some options below.



The first four terms are **economics**-biased. That's fine; it makes sense for analyzing price movement, valuations, and so on.

I'm trained as an electrical engineer (EE). EEs doing circuit design have theory, have practice, and build systems that *just work*, such as the screen you're reading and the chips that power it, to the lights over your head.

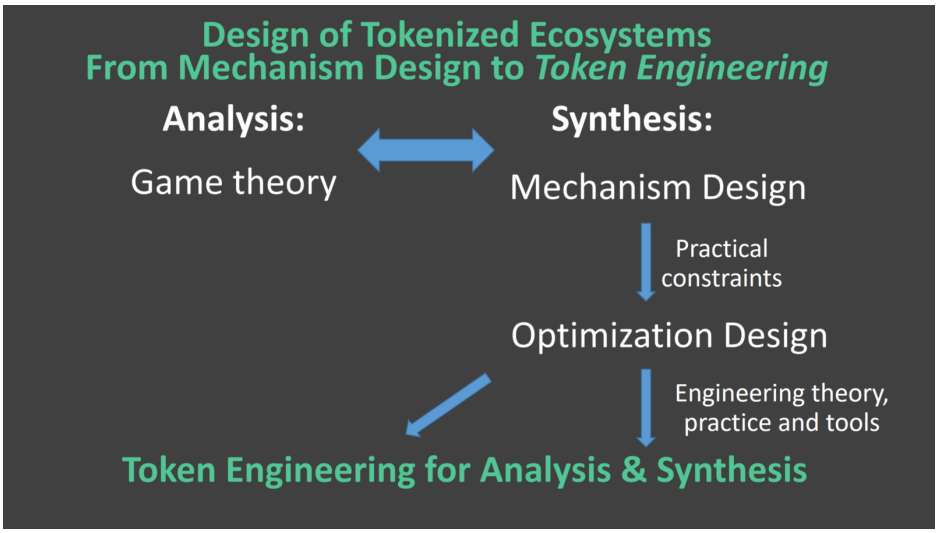
Engineering is about **rigorous analysis**, **design**, and **verification** of systems; all assisted by tools that reconcile theory with practice. Engineering is also a discipline of **responsibility**: being ethically and professionally accountable to the machines that you build, as illustrated by the Tacoma Narrows Bridge viewings and iron rings.

I saw the rise of the discipline of software *engineering* in the 90s; that made sense as it encouraged rigor and responsibility.

I'd love to see token ecosystem design become an engineering discipline, side-by-side with electrical engineering, software engineering, civil engineering, aerospace engineering, and so on. This implies that token ecosystem design would also become a field of rigorous analysis, design, and verification. It would have tools that reconcile theory with practice. It would be guided by a sense of responsibility. It would be **token engineering**.

[Note: As for "token" vs "incentive", "token" is shorter and easier to compare to its economics counterpart "Tokenomics".]

The image below shows how these fields relate. The goal is a practice of token engineering.

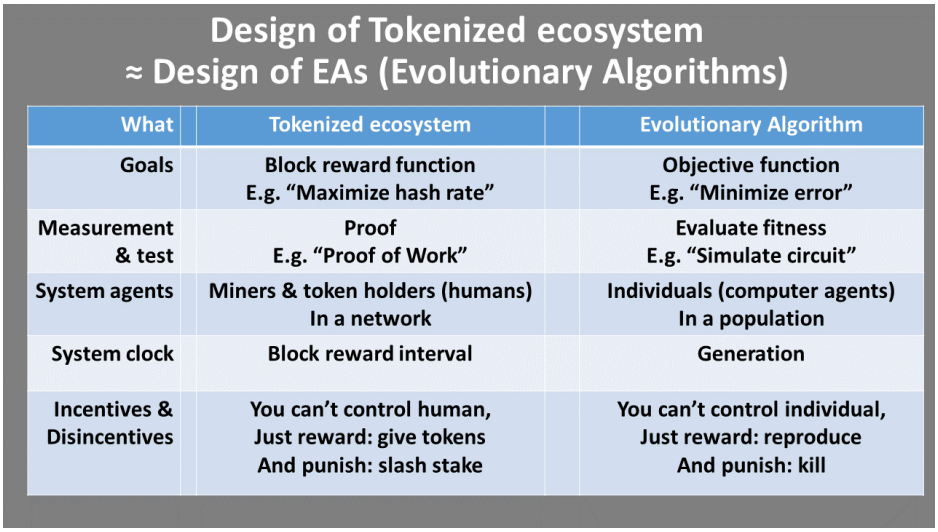


3. Token Design as Optimization Design

Token design is like optimization design: at a high level, you encode intent with a block rewards function aka objective function, and you let it fly. As is often the case, Simon de la Rouviere saw this one first.

It gets more specific than that. Token design is *especially* like evolutionary algorithms (EAs), where there are many agents "searching" at once and there is no top-down control of what each agent does. Agents live and die by their block reward or fitness. The table below summarizes the relation.

With such similarities, *we can use best practices from optimization / EA to when doing token design*. This is great news, because many people are Jedis in designing EAs and optimization systems.



Let's elaborate on each row in the table.

3.1 Goals

Both tokenized ecosystems and EAs have **goals**, in the form of **objectives** (things to maximize or minimize) and **constraints** (things that must be met). To get fancy, this can even be stochastic.

The tokenized ecosystem might give block rewards for an objective function of "maximize hash rate" whereas an EA objective might be "minimize error" in training a deep net. Constraints might be "must have stake ≥threshold to participate" or "deep network layers=100" respectively.

Variants include single-objective optimization (1 objective, 0 constraints), constraint satisfaction (0 objectives, ≥1 constraints), and multi-objective constrained optimization (≥2 objectives, ≥1 constraints).

3.2 Measurement & Test

To **test / measure** success against the goals (objectives & constraints), a tokenized ecosystem relies on **proofs** and an optimizer measures **fitness** using e.g. a **simulator**.

For example, a Bitcoin node proves that a user was hashing by verifying that the user's supplied nonce solves the the cryptographic puzzle.

An optimizer might test the goodness of a circuit by running a SPICE simulation of the circuit's differential equations; simulation results can be verified by testing whether they indeed solved Kirchoff's Current and Voltage Laws.

3.3 System Agents

In both systems, **agents** run about "doing things".

In a tokenized ecosystem, **network stakeholders** such as **miners (**or **users** more generally) do whatever it takes to earn block rewards. They jostle about, doing what it takes to get more token rewards. For example, in Bitcoin some agents might design, build, and run ASIC chips to get higher hash rate. Other agents might pool their existing compute resources. The system does not need to explicitly model all stakeholders in the ecosystem. For example, Bitcoin doesn't have specific roles for banks or nations or companies; it's mostly all about the miners.

In an EA, you have **individuals** in a **population.** If they're "good" they have higher fitness. For example, an individual may be a vector of 10,000 weights for a neural network. "Actions" of individuals are basically when they survive and have variants made of them, via operators like crossover (e.g. interpolation) or mutation (e.g. randomly perturbing each parameter).

3.4 System Clock

Each system has a **clock**, implying a **time dimension** by which **progress** is made / **convergence** is happening.

Batches. Typically, agents are processed in batches or epochs. A tokenized ecosystem periodically generates a **new block** and gives block rewards. The new block points to the old block; and new work in the system will add to the new block; and so on. This linked list of blocks implies a Lamport-style logical clock. In EAs, each batch is a **generation** where a whole population of individuals gets updated at once. Each

generational loop might include: evaluate individuals, select the best, let them make children, repeat.

Continuous. In some systems, agents are processed more **continuously** rather than batches. These systems usually takes a bit more work to conceptualize, but may lead to better properties for some problems. For example in tokenized ecosystems, a Stellar transaction only needs validation from quorum slice participants, or another node gets added to a DAG (directed acyclic graph) like in Iota. In EAs, we have steady-state evolution where one individual at a time is replaced.

3.5 Incentives & Disincentives

The system itself cannot control how the agents behave. (Or at the very least, it shouldn't *need* to control them.) As such, **top-level behavior must be an emergent property of bottom-up actions by agents**. This is necessary for tokenized ecosystems; otherwise they'd be centralized! It's not an absolute must for EAs, but nonetheless a broad set of EAs take this approach for simplicity / elegance or meeting other design goals.

This means the system can only reward or punish behavior, aka carrots or sticks, aka **incentives** and **disincentives**. In designing the system, we *design* what rewards or punishments to give, and how to give them.

In tokenized ecosystems, rewards take the form of block rewards, and punishments by slashing stake. The former is typically the objective function; the latter is some (but not all) constraints.

In EAs, reward and punishment both come down to which individuals are selected to be parents for the next generation. Examples: randomly choose two individuals and keep the best, repeating until full (tournament selection); and chance of selection is proportional to fitness (roulette wheel selection). Crucially, the EA does *not* need to steer the individuals by e.g. providing a derivative. This is why a tokenized ecosystem is most like an EA, versus gradient-based optimizers that give top-down directives (using gradients to choose new individuals).

4. From Optimization Methodology to Token Methodology

4.1 Introduction

This section is some initial notes towards treating token design like the engineering discipline it deserves to grow into.

First, I describe a structured methodology for optimization design. Then I describe how a similar methodology could be applied for incentive design.

Next, I describe key tools used in circuit design: simulators and CAD tools; and how they might be applied for circuit design.

Finally, I start to enumerate some design patterns.

4.2 Optimization Methodology

These communities only partly talk to each other. But practitioners of the algorithms all do something very similar. They want to ship optimizer systems that *just work*. They follow the following steps. Some do it implicitly, though the pros do it systematically:

1. **Formulate the problem:** They assume that the algorithm "just works" and they focus on formulating the problem in terms of **objectives and constraints** (goal) and design space (where can the optimizer explore, which is really just constraints).
2. **Try an existing solver:** Then they run the algorithm against those goals and let it "solve". Code for optimization algorithms are often simply called "**solvers**". If this doesn't work, practitioners will iterate by trying different problem formulations, or different solvers and solver parameters.
3. **New solver?** If the previous solving step doesn't work, even after repeated tries on various formulations, then practitioners consider rolling their own solver, i.e. designing a new optimization algorithm.

Let's explore each step in more detail.

Step 1. Formulate the Optimization Problem

Look in almost any optimization-related paper, and you'll see it display the objectives and constraints. In examples from my own work, equation (1) of this paper (in sec. 5) is a formulation for a multi-objective constrained optimization across a search space defined by a grammar. Here's a snippet:

> The algorithm's aim is formulated as a constrained multi-objective optimization problem
>
> $$\begin{array}{lll} \text{minimize} & f_i(\phi) & i = 1...N_f \\ \text{s.t.} & g_j(\phi) \leq 0 & j = 1...N_g \\ & h_k(\phi) = 0 & k = 1...N_h \\ & \phi \in \Phi & \end{array} \quad (1)$$
>
> where Φ is the "general" space of possible topologies and sizings. The algorithm traverses Φ to return a Pareto-optimal

As another example. Equations (1) and (2) of this paper (in sec. 2) lay out a stochastic single-objective ("maximize yield") optimization search problem across an n-dimensional continuous-valued variable space.

Formulating a problem in objectives, constraints, and design space is not easy. In fact, after all these years, it's still an art that takes a lot of creativity. (Which means it's fun!) There are often many ways to formulate a problem; and all are not equal . Fortunately, you can get better with practice. I watch friends in the EA community and circuit computer-aided design (CAD) community who have supreme skills in the art of formulating problems. You know who you are;)

- For example, one formulation might be easy to solve and another might be NP-hard and you can't guarantee anything. This is one of the big tricks used by people in the convex optimization field: they take problems that are perceived as NP-hard, then apply techniques to convert those to convex problems (e.g. well-placed log

operators). Then, these convex problems can be solved in polynomial time using a convex solver like geometric programming. I've found success in this too; for the problem summarized by the snippet above, I added grammatical constraints to a tree induction problem (analog circuit synthesis) and saw a 1000x improvement in runtime as well as better results from the optimizer.

- Or, if you're using an evolutionary algorithm (EA), you want to design the mapping from design point → fitness such that small changes to the design usually lead to small changes to behavior and ultimately fitness. (I call these smooth operators ;)

Step 2. Try an Existing Solver

Ideally you can formulate the problem such that you can apply an existing solver or algorithm. Then, you simply run it and see how it does. Hopefully, you're done, and you can stop now!

Step 3. New Solver? Design a New Optimization Algorithm, If Needed

Sometimes you come across a problem where none of the existing solvers are good enough. Perhaps you need better scale, or perhaps you need to handle some constraint that's hard to model, or perhaps something else. That's when you go and do research on algorithm design. When you're doing this, you'll usually leverage existing building blocks, and add in your own ideas. Designing new algorithms can take a tremendous amount of time, but done well, you may be able to solve the problem at hand with order-of-magnitude improvements, for example FFX.

(And, did I mention, it's *fun*?)

4.3 Token Methodology

Block rewards are a manifestation of the network's **objective function**—the thing you want to minimize or maximize. This generalizes. Token design is like design of optimization algorithms. Therefore:

We can approach token design as optimization design.

We can take the steps from the world of optimization design, and translate them into designing tokenized ecosystems.

1. **Formulate the Problem:** write down the objectives and constraints for your tokenized ecosystem. This means asking: who are my potential stakeholders, and what do each of them want? What are attack vectors? Then translating those into objectives and constraints that you can measure.
2. **Try An Existing Pattern:** Identify if there is an existing solver, i.e. tokenized network design pattern that can solve your problem. For example, if you're looking for a list of "good" actors/assets/etc, will a token curated registry (TCR) do? I elaborate on this later.
3. **New Pattern?** If needed, roll your own solver, i.e. design your own tokenized network. Of course when doing this, use existing building blocks where possible, from TCRs to arbitration.

5. Token Design Patterns : A Starting Point

Every mature engineering field has its corpus of building blocks .We have books for design patterns in architecture, software, analog circuits and, yes, optimizers. But, no one's written the book on *token design patterns*, yet.

However, building blocks are starting to emerge. Some have seen popularity explode quickly (e.g. TCRs). The paragraphs below explore these blocks. Sometimes they form core token mechanics; sometimes they get bolted in to solve particular problems. This list is just a starting point.

- **Curation.** *Binary* membership: Token Curated Registry (TCR), e.g. to maintain a list of good actors. *Discrete-valued* membership: Stake

Machines, e.g. for promoting an actor. *Continuous-valued* membership: Curation Markets(CM) for popularity of an asset. *Hierarchical* membership: each label gets a TCR (like here). *Work* tied to membership: Proofed Curation Market.

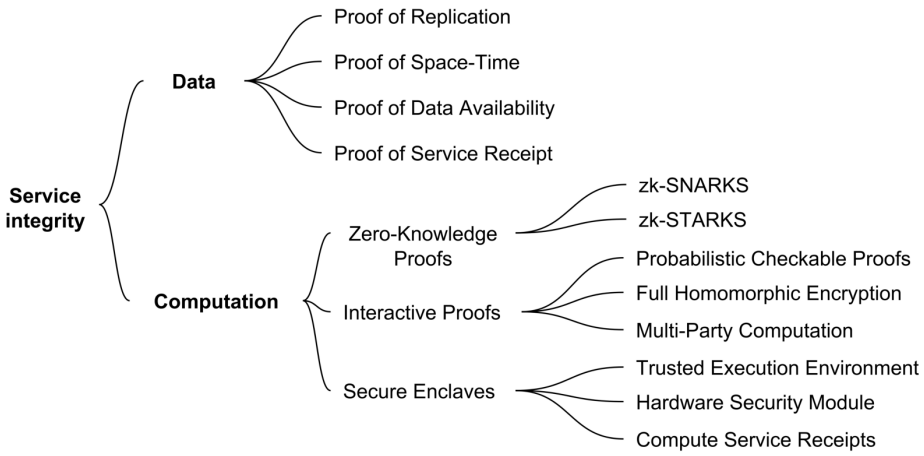
- **Identity.** *Lower level:* public key, decentralized identifiers (DIDs). *Medium level*: TCR. *Higher level*: e.g. uPort, Civic, Sovrin, Authenteq, Taqanu, Estonia E-Residency. *Identity of machines:* e.g. Spherity

- **Reputation.** Reputation systems are at the intersection of curation and identity.

- **Governance / software updates.** This can be a mix of ZeppelinOS, Aragon, Colony, and more.

- **Third-party arbitration.** E.g. Mattereum.

- **Proofs of human or compute "work".** For data, compute, and more. This the evaluation of the objective function. It can be can *human work* like in Steemit or Augur, or *machine work* like in most other systems. Machine work may be solving an (arguably) less-useful puzzle like in Bitcoin, or more "useful" work like FileCoin's Proof of Space-Time. Here's a breakdown of useful work ("service integrity") grouped by *data* and *computation* (from here).



Other ways to frame or group building blocks include:

- **How tokens are distributed.** This includes releasing coins for "work", according to a controlled supply schedule; 100% pre-mining; burn-and-mint; bounty ICOs; and more.
- **How tokens are valued.** As a means of exchange, store of value, and unit of account, by Chris Burniske.
- **How keepers are grouped.** For gatekeeping, arbitrage, or resource transaction, by Ryan Zurrer.
- **How the compute stack is organized.** Processing, storage, etc. This has variants by Fred Ehrsam, Stephan Tual, and myself.
- **Level-1, level-2, level-N infrastructure.** The core chain is level 1. The higher levels are to help scale without having to reconcile the main chain on every transaction. Link.

This enumeration of patterns is just a starting point; I look forward to watching it grow.

6. Tools We Need: Simulators, CAD Tools

6.1 What

Professional engineers building things that "just work" use *software tools* for it. Software engineers often use integrated design environments (IDEs) that are free or relatively cheap.

But you can get much more sophisticated. In circuit design, the key tools are simulators and CAD tools. The tool stacks can become quite sophisticated over time. But with these tools, it enables a team of 10 engineers to design a billion-transistor chip in a matter of *months*. A good engineer might be running $1M worth of tools (that's the *annual* licensing cost!).

> *"Now You're Playing with Power"—Nintendo 1980s marketing slogan*

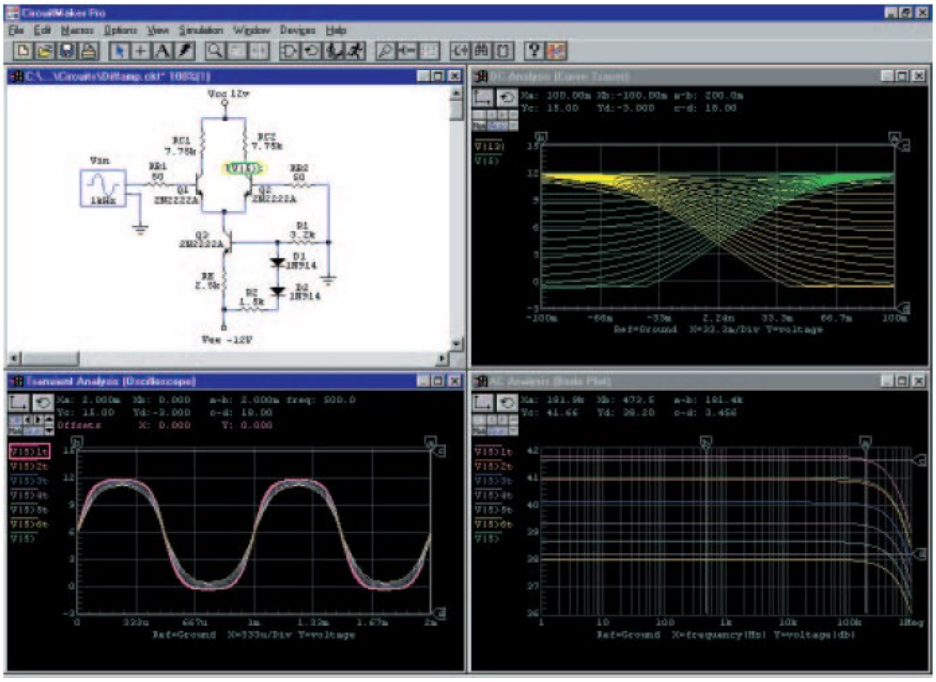
Let's get some power tools for this new field. We need:

1. Proper **simulators** of tokenized ecosystems. One starting point is the agent-based modeling that's coming out of the fields of complexity science and artificial general intelligence. Another is the simulators for networks already used for consensus algorithm design.

2. We need **CAD tools** that *wrap* the simulators to verify designs further, gain insight on how the variables and outputs react, and explore the design space.

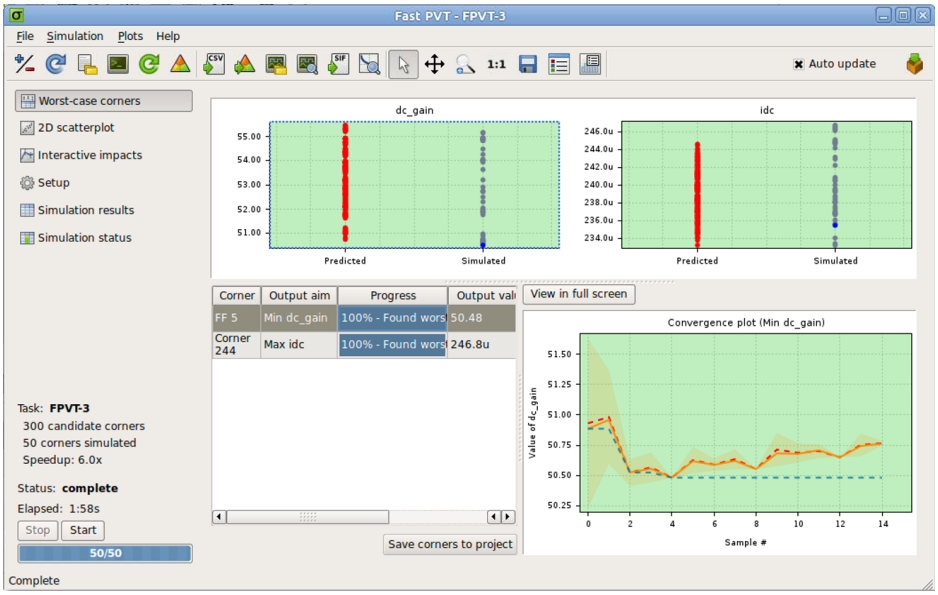
6.2 Simulator and CAD Tools from Circuit Design

The figure below shows an example of a circuit simulator environment. The top left is a schematic editor to inputting the design. For analog circuits this is the choice of resistors, capacitors, transistors etc; how they are connected; and what their sizes are. That input is then automatically converted to a set of differential equations that are then solved using the simulator. The other three windows show results of various simulations. Clockwise from top right are steady-state (dc) analysis, time-based (transient) analysis, and frequency (ac) analysis.

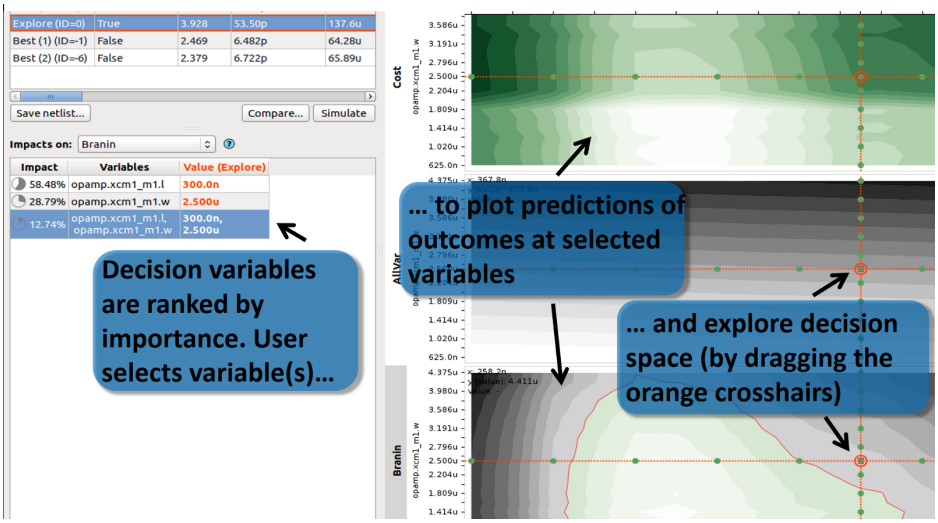


[Image: Wikimedia Commons]

The next two images are from CAD tools that I helped to develop. Below is a tool that **verifies** that the chip will not fail across a range of worst-case "PVT" conditions: extremes in **p**ower supply **v**oltage, **t**emperature, load, etc. Under the hood, it's got a global optimizer that tries to optimize towards failure.



The image below shows a tool to **explore** the design space. It reports relative impact of variables on various outputs (left), and how the design variables map to outputs (right). The engineer can change the designs by dragging the orange crosshairs.



These and other tools are now widely used to design modern chips. Simulators came on stream in the 1970s and CAD tools in the 1980s; and no one's looked back. These tools are *crucial* to modern chip design. It costs >$50M to manufacture a design on a modern process; it would be, well, *stupid* to not verify and optimize that design to the best possible level before committing the $50M.

Yet in the world of token design, we are building and deploying what we hope to be billion-dollar ecosystems, with barely any tools. It isn't even 1970 yet. I look forward to the day when we get to this level for token engineering!

7. Conclusion

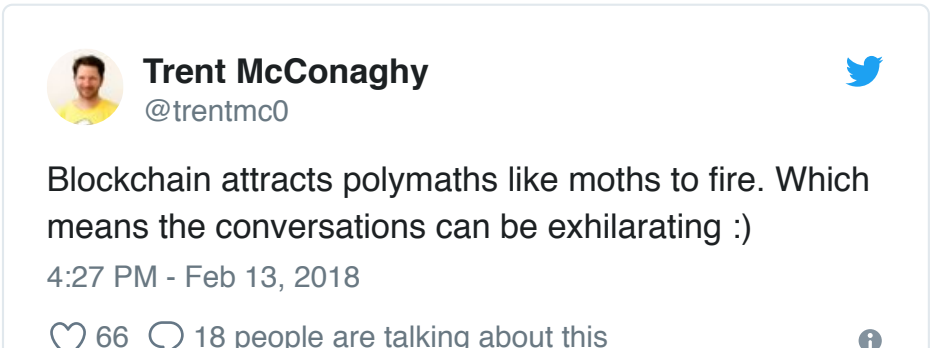
This article described how we can leverage existing fields to help design tokenized ecosystems: token design as optimization design; token design patterns; and token design tools inspired by circuit design tools. The overall goal is a practice of **token engineering**. We'll get there!

Appendix: From People to AIs

While this article has emphasized the use of block rewards to get *people* to do stuff, they can also get *AIs* to do stuff. There are interesting implications…

Appendix: Calling All Polymaths

I've noticed that there's a group of people in blockchain that seem to thrive especially well. It's the learning machines: the folks who learn for fun, build for fun, who dance among many fields and build bridges between them. Yep, the polymaths.



In this article, I've described how practices from optimization and other fields could help in designing tokenized ecosystems. It also means that experts from these fields could find their skills to be useful in the brave new world of blockchains. Furthermore, just like in blockchain: many folks from AI, complex systems and more are natural polymaths.

I've seen this first hand. My own experiences in AI and optimization have been extremely helpful to grok blockchain. Also, I've found it easier to ramp up AI people by teaching them the delta between what they know, and blockchain. I simply describe tokenized systems as EAs! To the Artificial Life people, it's life. To the electrical engineers, it's feedback control systems. And so forth.

Appendix: Related Articles & Media

This article is part of a series. I will link to the other articles as I release them:

- Part I. Can Blockchains Go Rogue? AI Whack-A-Mole, Incentive Machines, and Life.
- [This article] **Part II. Towards a Practice of Token Engineering:** Methodology, Patterns & Tools.
- **Part III. Token Engineering Case Studies:** Analysis of Bitcoin, Design of Ocean Protocol.

I gave a talk about much of this content in Berlin in Feb 2018. Here's the **slides** and **video.** I gave a related talk about complex systems at Santa Fe

Institute, New Mexico in Jan 2018. Here's the **slides** and **video** from that talk.

Acknowledgements

Thanks to the following people for reviews of this and other articles in the series: Ian Grigg, Alex Lange, Simon de la Rouviere, Dimitri de Jonghe, Luis Cuende, Ryan Selkis, Kyle Samani, and Bill Mydlowec. Thanks to many others for conversations that influenced this too, including Anish Mohammed, Richard Craib, Fred Ehrsam, David Krakauer, Troy McConaghy, Thomas Kolinko, Jesse Walden, Chris Burniske, and Ben Goertzel. And thanks to the entire blockchain community for providing a substrate that makes token design possible:)