Response to a proposed funding model (DWeb 2024)

The Problem Space 🎈

(purely from an outsider's perspective)

Issues the funding model seeks to address 🦄

- How to prevent private IP from becoming orphan works or abandonware
- How to scale and ensure both stable cash flows & ROI
- How to fix the many-headed hydra
- General coordination & administrative support for all the above

Issues left unaddressed 🪚

(again, purely as an outsider looking in)

Revenue 💸

If this plan doesn't provide a general mechanism for how the enterprise layer can generate revenue (presumably leaving that up to individual founders), then how can it actually provide any guarantees to investors, beyond the typical venture strategies (e.g., for every 99 that fail, 1 becomes a unicorn)?

Revenue for many existing projects, such as farmOS, is derived solely from labor time spent developing new features and maintaining the source code, but if this model does not propose alternative revenue streams, what value can developers expect in return for the interest paid out to investors, which presumably they would have otherwise kept for themselves?

Capital assets 🏦

Another possible way to provide some guarantee to investors, however...

- Is there a plan for how to produce or acquire them?
- Where will they come from and who will produce them?
- Can free software be considered as capital assets?
- If not, will existing free software be relicensed under non-free terms?

Other Concerns

(take with a big grain of salt)

Licensing & the Commons ©

- Is this model undervaluing the role of independent contributors in the free software community and the generative effects if provides?
- Does it risk jettisoning that value and estranging independent contributors if this plan stifles or significantly alters those community dynamics?

Costs of complexity 🫠

How can the utility's funding pool simultaneously:

- Pay investors capped returns for existing free software projects
- Buy out private IP from exiting founders
- Cover administrative and operating costs of the utility organization's board and/or staff
- Leave any profit for the individual enterprises*

* That is, if we assume some of their revenue is reinvested into the funding pool.

Need for complexity 🤔

- Do we really need all this?
- What about unforeseen costs?
- How and when would we know if that cost is too great?
- Can we afford the *risk* of this complexity?

Complexity: essential or accidental 🪃

If we assume a need to pay investors, or acquire private IP?

- Then yes, it may be essential.

If we don't need any of that?

- I'll wager it's incidental (and avoidable).

Runrig 🚜

A method of socio-ecological design 🌱

Runrig's problem space

What Runrig does **not** address:

- Salvaging orphan works & abandonware
- Guarantee of ROI to past & future investors

What Runrig *does* address and hopes to provide:

- A revenue model that provides stable income to developers and affordable prices to food producers
- A means of surmounting local maxima in the fitness landscape of free software development (aka, the "many-headed hydra")

Runrig's general strategy

- Distribute **costs** and **control**
  - *contra* capital accumulation & marketing
- Organize **support** and **reach**
  - *contra* scaling & infinite growth

And overall, it aims to start with the 🍪's — preferably many small ones — then build incrementally towards the great big 👾.

A Three-Phase Organizing Model

- *Phase One:* **Middleware, a.k.a. "glue services"** 🍯
- *Phase Two:* **The One-stop Platform Co-op Shop** 🛒
- *Phase Three:* **One Big Data Co-op** 🐈‍⬛

Phase One: Middleware, a.k.a. "glue services" 🍯

Targeted interventions that catch the spillover from other FOSS projects' `wontfix` issues; e.g.,

- API integrations
- extensions, plugins, middleware, etc.
- lightweight local-first applications
- component libraries
- client libraries
- and so on...

Phase Two: The One-stop Platform Co-op Shop 🛒

- Platform bundles together & hosts complementary **applications, middlewares & services** appropriate to and customized for that community
- Services may be specific to food and agriculture like farmOS or Brinjel, or more generic services like Nextcloud or Baserow
- Single Sign-On (SSO) & other niceties reduce complexity for end-users.
- Full control + ownership belongs to the community: farmers, cooperatives, buyers clubs, individual consumers, and related small businesses
- Worker co-op(s) of developers, designers, & TSPs are paid to develop, maintain, & provide varying levels of support for the platforms.

Phase Three: One Big Data Co-op 🐈‍⬛

- A global data cooperative or (better yet) a **data trust**
- Communally owned and controlled by its users
- Fills the role of **data provider** to the service platforms collectively
- Members comprises 2 classes:
  - **Data Members:** Individuals, farms, etc. who store their data
  - **Service Members:** The service platforms themselves
- Leverages semantic data and standards to enable greater interop and relieve the computation & storage burden on platforms & local devices, *plus a whole lot more details at runrig.org*.

Case Study: Farm Flow

A Runrig pilot application currently in development.

- Based on the original concept developed by farmers in Minnesota.
- Adapted from a physical whiteboard for coordinating the execution of SOPs in accordance to a specialized regimen for transitioning to organic (also the farmers' original creation).

Farm Flow: Development Cycle

- ~$50k of combined grant funding & farmer contributions
- Slated for launch Winter 2024-25
- Deliverables
    - Low Fidelity Pilot *(completed Jul 2024)*
    - Backend Server Integration *(est. completion by Nov 2024)*
    - High Fidelity Pilot *(est. completion by Feb 2025)*

Deliverable #1: Low Fidelity Pilot

Implemented a local-first application for the browser using IndexedDB and cheaply/freely hosted on a CDN. Started with the data model in serialized JSON that can be imported and exported from its very first iteration, before connecting to a server. *(completed Jul 2024)*

Deliverable #2: Backend Server Integration

Bring in farmOS and/or LiteFarm as the backend to provide synchronization and mobile data capture. *(est. completion by Nov 2024)*

Deliverable #3: High Fidelity Pilot

With feedback from users and collaborating free software projects, add the necessary feature enhancements and polish the user experience for the final production deployment. *(est. completion by Feb 2025)*

So what now?

- Can these two models co-exist? (I think yes)
- Can they be mutually supportive? (also yes)
- What do they have to learn from each other? 🤔
- 🦄 + 🚜 = 🪄 ?