Recent Progress

Evolvix architecture development has been focusing on developing foundational technologies for simplifying compiler construction in the presence of many moving components and their names that can be in flux for some time due to the user input invited by the *Design Flip* (see figure on other page). Testing this concept extensively has shown that stabilization is possible and produces high-quality results. This is enabled by important yet unassuming innovations that have been the initial focus of development and will continue to grow as needed (see Evolvix.org/concept):

- Stabilizing Versioning to improve system stability
- Automated resolution of synonymous names
- Better batching strategies created by the *Design Flip,* greatly improving review quality and speed.
- Systematic project organization with the POST system (includes a StabilizingZone).

Where Next?

Evolvix architecture development continues to focus on foundations. Like the pillars of buildings, these are key for stability. Redefining core concepts is the programming language equivalent of blowing up a building: best done on paper if possible and before publication!

Availability may take a while ...

... and that is by design! Instead of wasting the time of users by shipping unstable designs, Evolvix makes a point of shipping only code that survived repeated rounds of rigorous review from relevant perspectives. Evolvix would rather disappoints on shipping dates than risk being less stable, extensible, and user-friendly over the long term. Important last minute discoveries can require such delays.

Prototypes. Like all developers, we try things. Some tests require implementation and some of these are shared on Evolvix.org for short-term use by experts; but they are instable; their syntax can change any time. To manage code maturing between these extremes, Evolvix uses its Stabilizing Versioning System (StabVS) whenever possible; see **evolvix.org/versioning**

Evolvix Statements Flyer PPv3r0p0 2018-10-09
by Laurence 'Lion' Loewe, Architect of Evolvix
Contact email: Laurence.Loewe@gmail.com

Current & Past Supporters (not responsible for any statements on this flyer):
National Science Foundation Program Advances in Biological Informatics,
Wisconsin Institute for Discovery, Laboratory of Genetics, UW-Madison.

Vision

Evolvix improves decision-making by modeling floods of data, conflicting values, and assumed logics, chronicling decision history, observed outcomes, and human annotations.

Methods

How?

Integrate Expertise by *pre*factoring and participatory design together with the **evolvix.org/thinkers**

Want to contribute? Contact the Architect if you would like to volunteer as *usability* or *expert reviewer* and would like to discuss various aspects of biodata science or how its concepts could be expressed in an elegant syntax for Evolvix.

Why?

purpose analogy: **Evolvix** is about building **life rafts** that **save people from drowning in floods of data**

Evolvix

Mission

Simplify Accurate Modeling of systems described in a Stable Extensible User-friendly Language

How Evolvix started is a long story...

and includes the Lion's share of a two decades full-time career in computational biology that now informs Evolvix design. Here's a way to summarize it: Evolvix started to crystallize first as a domain-specific modeling language for accelerating realistic model construction for biologists, aiming (i) to **make rigorous modeling from molecular systems biology available for ecology and evolution**; and (ii) to offer a stable extensible user-friendly syntax and simulation environment for **encouraging beginners to learn and professionals to perfect their modeling skills**. It aims to use best-of-breed algorithms to analyze diverse dynamical systems, reporting results efficiently. The idea won a 2012 NSF Career Award. 2013 saw Prototype 0.1, 2014 the first research paper[1], 2015 the better Prototype 0.3.1 and a prototypic web interface. **Evolvix is good at observing time series in pure mass-action models.** Since 2013 it is used each year in the evolutionary systems biology course and the Lion's research lab to build many diverse models. Such real-world use is challenging and has been ruthlessly revealing many deficiencies. A big one showed while plotting time series for the research paper[1]: **User-friendly time series analyses depend on general-purpose programming paradigms**. Hence, strategies for Evolvix architecture development were adjusted to **enable inclusion of new programming paradigms,** diverse modeling formalisms, and flexible analysis capabilities. This could make **Evolvix the first general-purpose programming language designed by biologists for biologists facing complex biology**. Experience shows that biology cannot afford to ignore a proven approach for 'slicing complexity'. Eventually, biology tends to need it's insights, often in contexts depending on other paradigms. Thus, a stable, elegant, and efficient **integration via well-designed approaches for sharing data among many diverse programming paradigms is pivotal** for a general-purpose language targeting biology; so is the use of diverse types of logic for describing imperfect biodata; so is the need to *be very user-friendly and reliable to invite* busy non-computing biologists to trust it with their data. Impossible? Indeed – this mountain's vertical rock face from base to summit is too dangerous for standard approaches like 'hiking'! The collapsing Tacoma Narrows bridge is often used to illustrate that 'wicked' software problems can only be *solved* after being 'solved'. By comparison, a general language for biology designed by usual approaches is like a network of Tacoma bridges: failing under real-world conditions that require re-design and re-implementation, only to fail from the next condition! Huge teams can build many work-arounds, but fixing core design problems of a language is not easily distributed.

Game over? No! – This is the most exciting result of the last 5 years of research on Evolvix foundations!

1. Ehlert & Loewe (2014). "Lazy Updating of hubs can enable more realistic models by speeding up stochastic simulations." J Chem Phys 141(20): 204109. - 2. Loewe et al. (2017) "Evolvix BEST Names for semantic reproducibility across code2brain interfaces" Ann NYAS 1387(1): 124-144; see also http://evolvix.org/naming - 3. Biodata science: Scheuer et al. (2017) "FlyClockbase." BioRxiv https://doi.org/10.1101/099192

But how? The many tools that build Evolvix ...

are their own long story and build on the Lion's share of 20+ years of full-time research in computational biology. These define many requirements, offer many solutions, and are leveraged by newer expertise in designing compilers, data stores, memory structures, algorithms, parsers, and the use of many diverse insights from ~16 key disciplines. But: **the core foundational technologies were developed and tested[2] for Evolvix** in the Lion's lab: the *Design Flip* (Fig.**A→B**); BEST naming; a stabilizing versioning system; a project org system. Combined, they reveal **a less obvious, much gentler slope at the back of the mountain that can be climbed with patience and a few simple tools** by those those who don't mind detours or fields debris cluttered – to stay in the analogy!

... and why it depends on biodata science!

Defining how to store simulation data is key for efficiency in Evolvix, a task seemingly well-defined at first. Yet, **inferring model parameters requires real-world biodata in the same format to avoid a host of conversion problems. Thus, we need a solution for storing biological observations too!** But biodata is imperfect and uncertain at many levels. Accurate capture needs progress on hard issues in logic, statistics, & epistemology –coupled with bio-expertise! To this end an **interdisciplinary compiler-constructions workflow** was built on the *Design Flip* (below) and tested successfully[3].

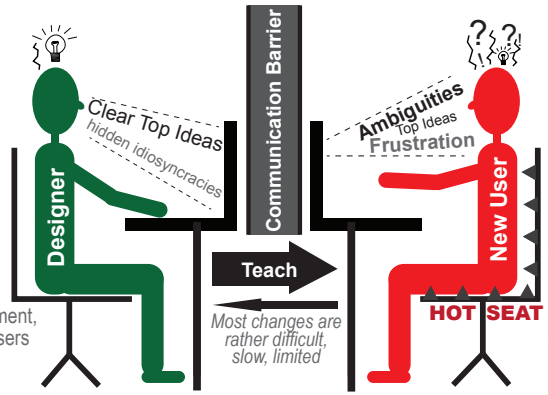
A. Usual Programming Language Design

Life as Designer ...

Easier to Design
Harder to Evolve

The designer accepts some avoidable complexity, does not see it as a core problem, or is unaware of how it can slowly poison a language.

Avoidable complexity creeps in with each ambiguity, idiosyncracy, oversimplification, missing key feature, confusing name, extra rule, legacy requirement, or other nuisance that expects users to remember or do things they do not need for their research.



Life as User ...

Harder to Learn

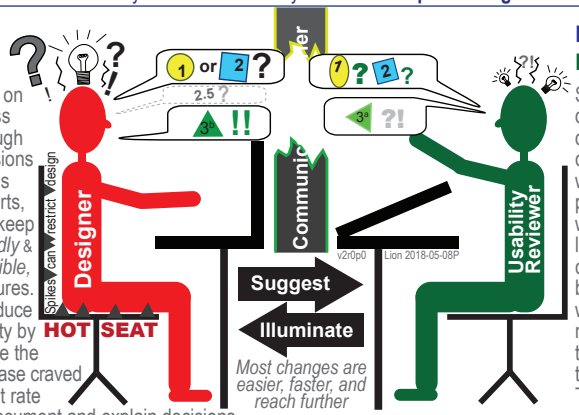
New users are hurled into the Hot Seat when they have to struggle with complexity that designers could have avoided by asking users. Facing avoidable complexity is usually frustrating and maybe even overwhelming; some users never recover and withdraw. Spikes symbolize rules that unexpectedly hurt users or delete data when ignored.

A design flip aims to avoid that learners have to bear the brunt of avoidable complexity in a language.

Flipped language design **asks users** how they would use a new syntax **before implementing** it.

Harder to Design
Easier to Evolve

The designer decides to take on the Hot Seat: find and address all avoidable complexity through sufficiently many design revisions based on in-depth discussions with usability reviewers, experts, and new users. The aim is to keep the language most *user-friendly & long-term backwards compatible,* while responsibly adding features. A designer's main job is to reduce long-term language complexity by strategic choices that combine the features for experts with the ease craved by beginners. Designers must rate feature stability accurately, document and explain decisions, organize review discussions, recruit reviewers, prioritize, etc.



Discuss to Design
Easier to Learn

Some potential users enjoy discussing feature and syntax design in advance. These users can excel as usability reviewers, who catch avoidable complexity prior to implementation. Their work is pivotal for reducing the long-term poison of avoidable complexity, which can quickly become invisible to a designer who often fails to predict where new users struggle. To reduce this blindspot, users share how they may interpret a given syntax. The designer takes it from there.

B. Flipped Programming Language Design

Takes a bit longer.
Lasts a lot longer.