# Caring for your environment(s)

Konrad Hinsen

Centre de Biophysique Moléculaire, Orléans, France
and
Synchrotron SOLEIL, Saint Aubin, France

8 November 2023

## An environment is...

### Merriam-Webster

1 the circumstances, objects, or conditions by which one is surrounded

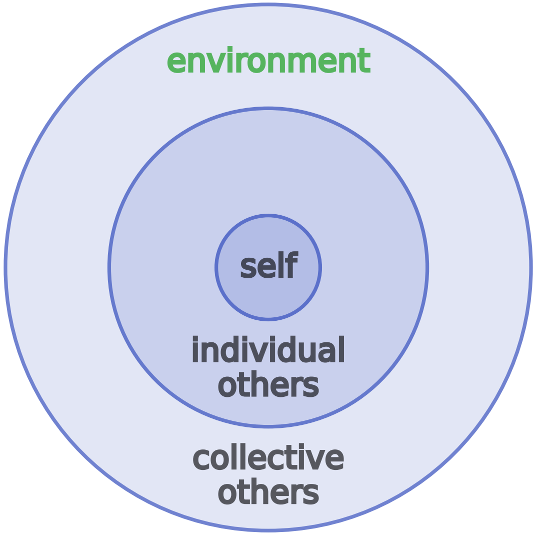
2a the complex of physical, chemical, and biotic factors (such as climate, soil, and living things) that act upon an organism or an ecological community and ultimately determine its form and survival

2b the aggregate of social and cultural conditions that influence the life of an individual or community

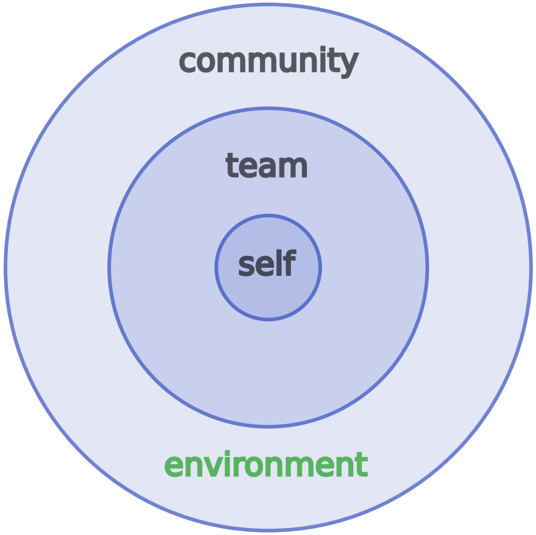
3 the position or characteristic position of a linguistic element in a sequence

4 a computer interface from which various tasks can be performed (ex.: a programming environment)

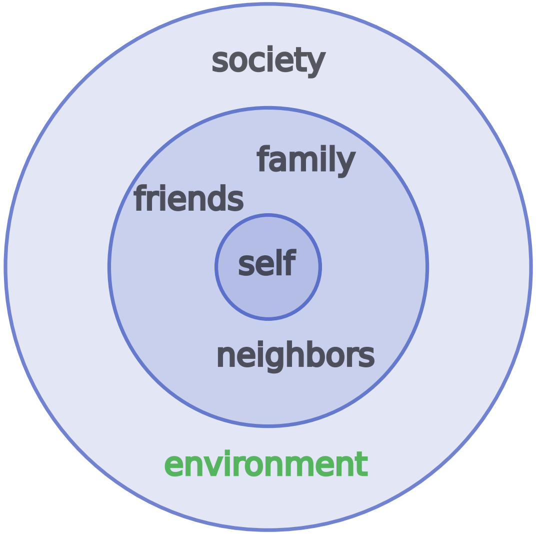
## Three spheres



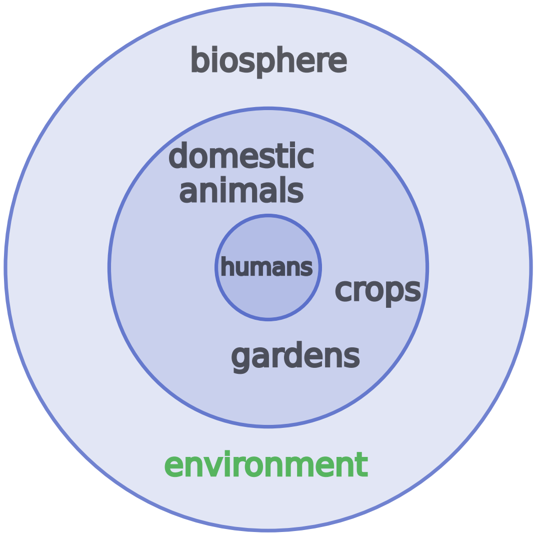
## Three spheres: work



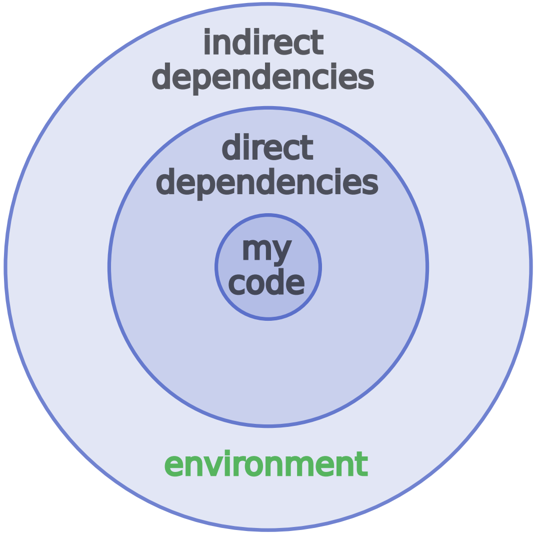
## Three spheres: society



## Three spheres: nature



## Three spheres: computation



## Caring **about** your environments

- Environments impact you and your work

## Caring **about** your environments

- Environments impact you and your work
- This impact tends to be **profound** but **invisible**

## Caring **about** your environments

- Environments impact you and your work
- This impact tends to be **profound** but **invisible**
- You should care about your environments

## Caring **about** your environments

- Environments impact you and your work
- This impact tends to be **profound** but **invisible**
- You should care about your environments

### Examples

- Communities define the roles in teams.
- Society makes the laws and conventions that govern your relations with family, friends, and neighbors.
- Compilers and interpreters define the semantics of your code.

## Caring **for** your environments

- What you do impacts your environments

## Caring **for** your environments

- What you do impacts your environments
- ... and other people’s environments!

## Caring **for** your environments

- What you do impacts your environments
- ... and other people's environments!
- This impact grows with the size of the team/community you are part of.

## Caring **for** your environments

- What you do impacts your environments
- ... and other people’s environments!
- This impact grows with the size of the team/community you are part of.
- You should care for your environments

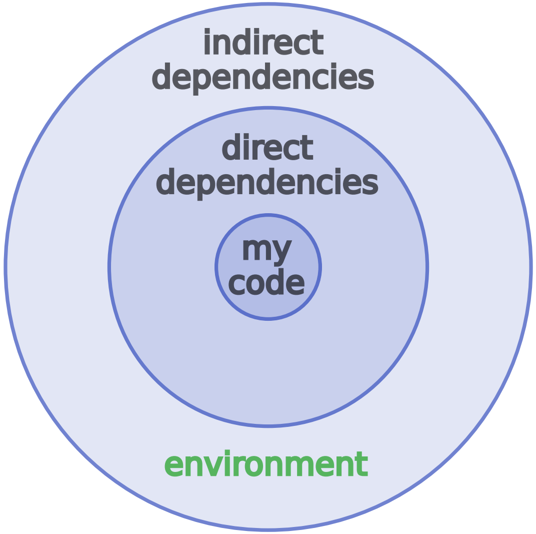
## Caring **for** your environments

- What you do impacts your environments
- ... and other people's environments!
- This impact grows with the size of the team/community you are part of.
- You should care for your environments

### Examples

- Your actions and behavior in a community inspire others.
- ... but also constrain others.
- Your adoption of a library increases its mindshare.

## Three spheres: computation



## Computational environments

## are mostly software foundations

## Computational environments in scientific computing

| | |
|---|---|
| **Project-specific code** | ***Scripts, notebooks, workflows, ...*** |
| **Domain-specific tools** | ***GROMACS, MMTK, ...*** |
| **Scientific infrastructure** | ***BLAS, HDF5, SciPy, ...*** |
| **Non-scientific infrastructure** | ***gcc, Python, ...*** |
| **Operating system** | ***GNU/Linux, ...*** |
| **Hardware** | ***x86 processor ...*** |

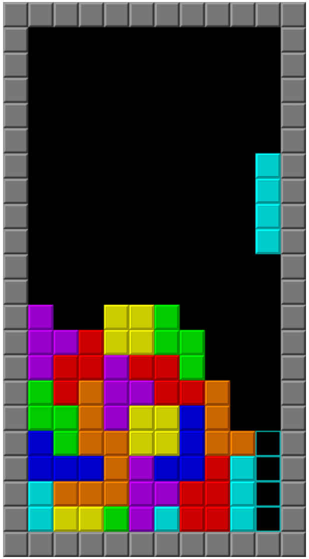
## Computational environments in scientific computing

| | | |
|---|---|---|
| Project-specific code | *Scripts, notebooks, workflows, ...* | my code |
| Domain-specific tools | *GROMACS, MMTK, ...* | my tools |
| Scientific infrastructure | *BLAS, HDF5, SciPy, ...* | |
| Non-scientific infrastructure | *gcc, Python, ...* | my environment |
| Operating system | *GNU/Linux, ...* | |
| Hardware | *x86 processor ...* | |

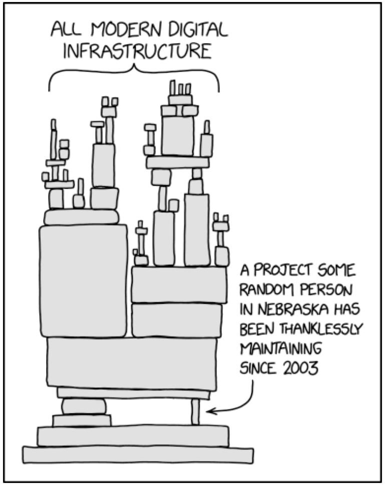
# Computational environments in scientific computing

| | | |
|---|---|---|
| **Project-specific code** | *Scripts, notebooks, workflows, ...* | **my code** |
| **Domain-specific tools** | *GROMACS, MMTK, ...* | **my tools** |
| **Scientific infrastructure** | *BLAS, HDF5, SciPy, ...* | |
| **Non-scientific infrastructure** | *gcc, Python, ...* | **my** |
| **Operating system** | *GNU/Linux, ...* | **~~environment~~ foundation** |
| **Hardware** | *x86 processor ...* | |

## A more detailed picture



## Risk of collapse


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

https://xkcd.com/2347/

## What’s the result of this program?

data_analysis.py

```python
from datalib import Dataset

points = [(1, 1), (-1, 1), (2, 4)]

data = Dataset()
for x, y in points:
    if x > 0:
        data.add_value(y)
print(data.average())
```

## What’s the result of this program?

data_analysis.py

```
from datalib import Dataset

points = [(1, 1), (−1, 1), (2, 4)]

data = Dataset()
for x, y in points:
    if x > 0:
        data.add_value(y)
print(data.average())
```

Quick answer: The average of $y$ for the points with positive $x \rightarrow$ **2.5**.

## What’s the result of this program?

data_analysis.py

```
from datalib import Dataset

points = [(1, 1), (−1, 1), (2, 4)]

data = Dataset()
for x, y in points:
    if x > 0:
        data.add_value(y)
print(data.average())
```

Correct answer: **The result depends on datalib**

## Know your libraries

datalib.py

```
class Dataset(object):

    def __init__(self):
        self.values = []

    def add_value(self, value):
        self.values = [value]

    def average(self):
        return sum(self.values, 0)/len(self.values)
```

Surprise! A bug! `add_value` stores only the last value!

The result of `data_analysis.py` is thus **4**.

## Know your libraries *and languages*

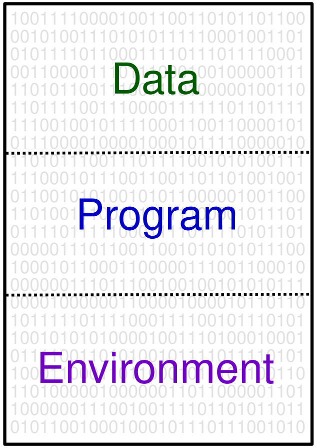
datalib.py

```
class Dataset(object):

    def __init__(self):
        self.values = []

    def add_value(self, value):
        self.values = [value]

    def average(self):
        return sum(self.values, 0)/len(self.values)
```

Surprise! A bug! add_value stores only the last value!

More precisely: **4** in Python 2 but **4.0** in Python 3.

## The meaning of bits



zeros and ones

interpretation of the data

interpretation of the program

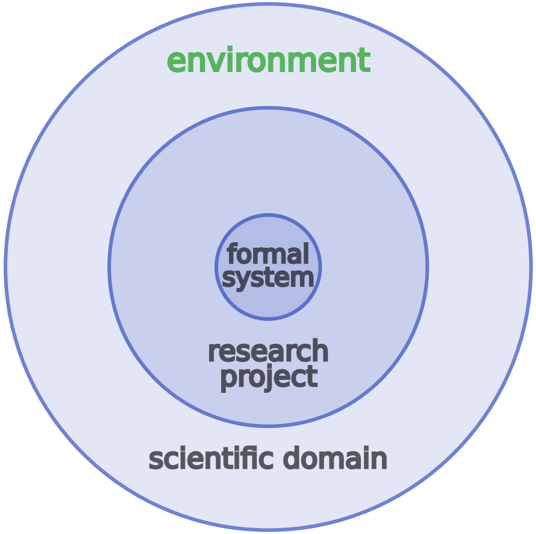
## Non-foundational parts of computational environments

- Network resources
  - Servers and their contents / services
  - Certificates
  - Identities

## Non-foundational parts of computational environments

- Network resources
  - Servers and their contents / services
  - Certificates
  - Identities
- Devices (e.g. physical random-number generators)

## The scientific environment of a computation



## Formal systems

- Mechanical manipulation of **symbols** according to **fixed rules**
- Symbols in – symbols out: **no interpretation**

## Formal systems

- Mechanical manipulation of **symbols** according to **fixed rules**
- Symbols in – symbols out: **no interpretation**

Examples:

- 16-bit strings with integer arithmetic

## Formal systems

- Mechanical manipulation of **symbols** according to **fixed rules**
- Symbols in – symbols out: **no interpretation**

Examples:

- 16-bit strings with integer arithmetic
- IEEE floating-point

## Formal systems

- Mechanical manipulation of **symbols** according to **fixed rules**
- Symbols in – symbols out: **no interpretation**

Examples:

- 16-bit strings with integer arithmetic
- IEEE floating-point
- Processor instruction set and data types

## Formal systems

- Mechanical manipulation of **symbols** according to **fixed rules**
- Symbols in – symbols out: **no interpretation**

Examples:

- 16-bit strings with integer arithmetic
- IEEE floating-point
- Processor instruction set and data types
- Programming languages

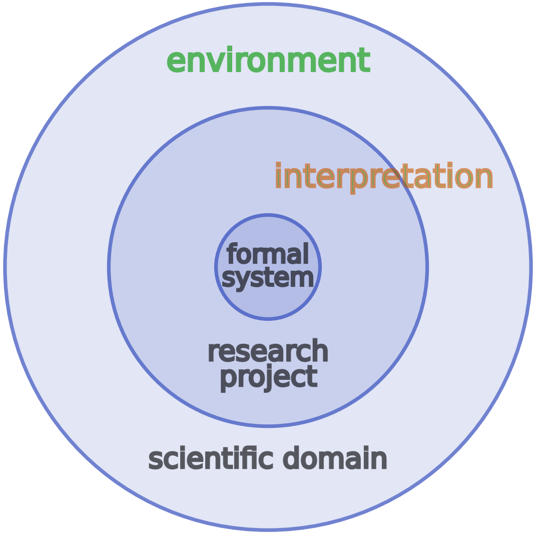
## Formal systems

- Mechanical manipulation of **symbols** according to **fixed rules**
- Symbols in – symbols out: **no interpretation**

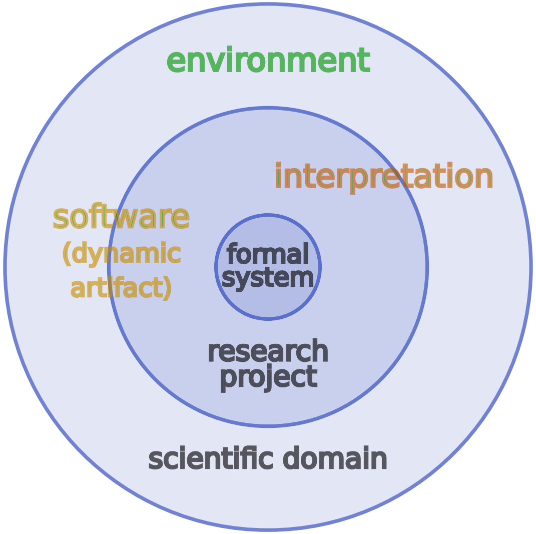
Examples:

- 16-bit strings with integer arithmetic
- IEEE floating-point
- Processor instruction set and data types
- Programming languages
- **Any program run on a computer**

## The scientific environment of a computation



## The scientific environment of a computation



## Three perspectives on scientific software

### Tool for doing science

- Evolves with its scientific environment
- Evolves with its computational environment

## Three perspectives on scientific software

### Tool for doing science

- Evolves with its scientific environment
- Evolves with its computational environment

### Expression of scientific knowledge

- Evolves with its scientific environment
- Should **not** depend on computational environments

## Three perspectives on scientific software

### Tool for doing science

- Evolves with its scientific environment
- Evolves with its computational environment

### Expression of scientific knowledge

- Evolves with its scientific environment
- Should **not** depend on computational environments

### Ingredient of a computation

- Fixed rules must remain fixed
- Must be reproducible

## Three perspectives on scientific software

### Tool for doing science

- Evolves with its scientific environment
- Evolves with its computational environment

### Expression of scientific knowledge

- Evolves with its scientific environment
- Should **not** depend on computational environments

### Ingredient of a computation

- Fixed rules must remain fixed
- Must be reproducible

Computations use **snapshots** of software.

## Three perspectives on scientific software

### Tool for doing science

- Evolves with its scientific environment
- Evolves with its computational environment

### Expression of scientific knowledge

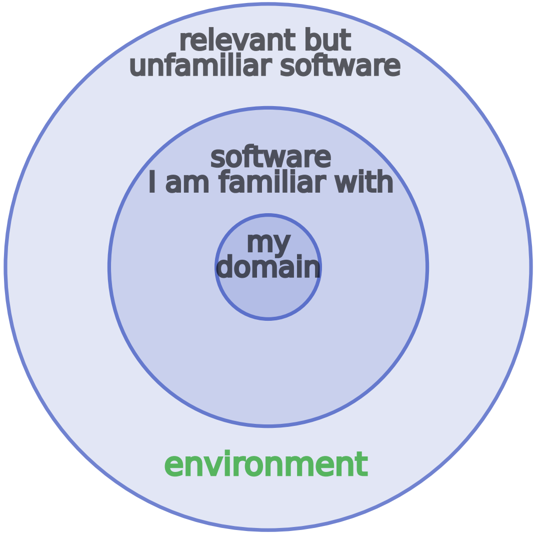
- Evolves with its scientific environment
- Should **not** depend on computational environments

### Ingredient of a computation

- Fixed rules must remain fixed
- Must be reproducible

Computations use **reproducible** **snapshots** of software.

## Three spheres: software



## Tool environments

We shape our tools,
and then our tools shape us.

Marshall McLuhan

## Tool environments

We shape our tools,
and then our tools shape us.

~~*Marshall McLuhan*~~ John Culkin (1967)

## Technical and social environments are related

**Melvin E. Conway (1967)**

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization’s communication structure.

## Technical and social environments are related

**Melvin E. Conway (1967)**

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

**Ivan Illich (1973)**

"Tools for conviviality"

- **Convivial tools** empower their users
- **Radical monopolies** make users dependent on a technocratic elite

## Sustainability

**Ecology**

Unlimited growth is not sustainable

## Sustainability

**Ecology**

Unlimited growth is not sustainable

**Scientific software**

Underfunded/understaffed projects are not sustainable

## Sustainability

### Ecology

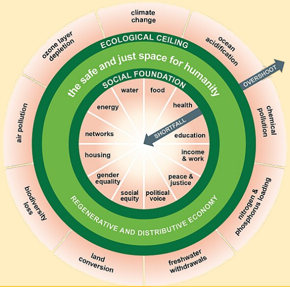
Unlimited growth is not sustainable

### Scientific software

Underfunded/understaffed projects are not sustainable

### Doughnut economy (Kate Raworth)

"A healthy economy should be designed to thrive, not grow."



Drawing by DoughnutEconomics

Own work, CC BY-SA 4.0

https://commons.wikimedia.org/w/index.php?curid=75695171

## Is our software project environmentally responsible?

Questions to (re)consider regularly:

- Are we empowering our users?
- What’s our user/developer ratio?
- Are we part of some users’ environments?
- Are our dependencies sustainable?
- Will our code be good legacy software?

## Is our software project environmentally responsible?

Questions to (re)consider regularly:

- Are we empowering our users?
- What's our user/developer ratio?
- Are we part of some users' environments?
- Are our dependencies sustainable?
- Will our code be good legacy software?

**Industrial mindset**

- growth
- efficiency
- competition

$\rightarrow$

**Ecological mindset**

- sustainability
- resilience
- coopetition

## A cautionary tale: scientific Python

**1995: The Matrix-SIG (Numerical Python)**

Let’s empower scientists to write their own code!

## A cautionary tale: scientific Python

**1995: The Matrix-SIG (Numerical Python)**

Let’s empower scientists to write their own code!

**2000: NumPy/SciPy**

Let’s win over Matlab users! Let’s *grow*!

## A cautionary tale: scientific Python

**1995: The Matrix-SIG** (Numerical Python)

Let’s empower scientists to write their own code!

**2000: NumPy/SciPy**

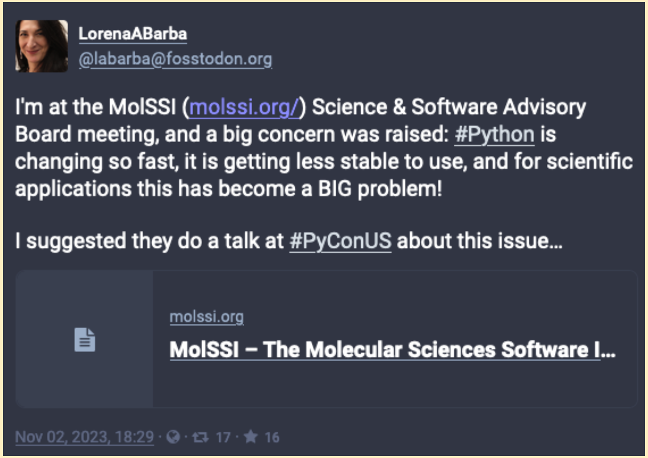
Let’s win over Matlab users! Let’s *grow*!

**2015: Scientific Python “ecosystem”**

Look, Google and Facebook are building on our code!

## A cautionary tale: scientific Python

### 2023: Scientific Python projects collapse



LorenaABarba
@labarba@fosstodon.org

I'm at the MolSSI (molssi.org/) Science & Software Advisory Board meeting, and a big concern was raised: #Python is changing so fast, it is getting less stable to use, and for scientific applications this has become a BIG problem!

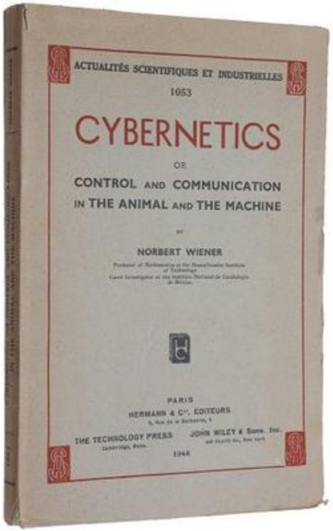
I suggested they do a talk at #PyConUS about this issue...

molssi.org
**MolSSI – The Molecular Sciences Software I...**

Nov 02, 2023, 18:29 · 17 · 16

## Outlook: what’s next in reproducible environments?

# Cybernetics



ACTUALITÉS SCIENTIFIQUES ET INDUSTRIELLES
1053

CYBERNETICS

OR

CONTROL AND COMMUNICATION
IN THE ANIMAL AND THE MACHINE

BY

NORBERT WIENER

[illegible]

PARIS
HERMANN & C^{ie}, EDITEURS
[illegible]

THE TECHNOLOGY PRESS
[illegible]

JOHN WILEY & Sons, Inc.
[illegible]

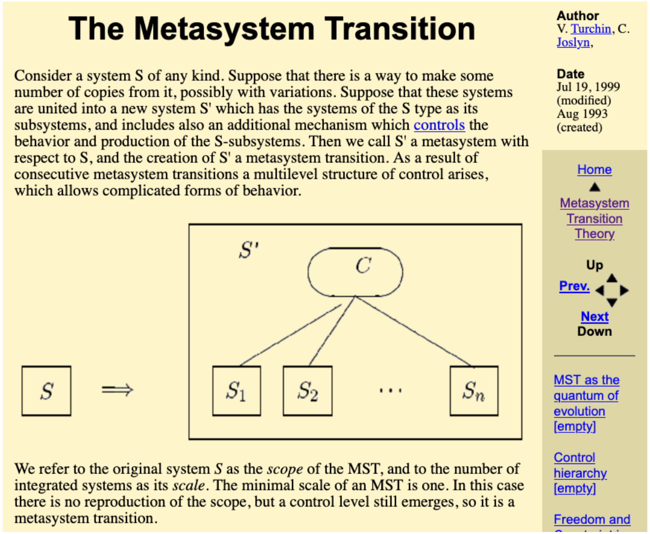
1948

Cybernetics
Or Control and Communication
in the Animal and the Machine
by Norbert Wiener (1948)

## Building computational environments

- 1970s: Install package after package by hand
  `./configure; make install`
- 1990s: Add package after package via support software
  `apt install gcc`
- 2010s: Define an environment and have it built by support software
  `guix shell -C -m manifest.scm`

# Metasystem transition



## The Metasystem Transition

**Author**
V. Turchin, C. Joslyn,

**Date**
Jul 19, 1999 (modified)
Aug 1993 (created)

Consider a system S of any kind. Suppose that there is a way to make some number of copies from it, possibly with variations. Suppose that these systems are united into a new system S' which has the systems of the S type as its subsystems, and includes also an additional mechanism which controls the behavior and production of the S-subsystems. Then we call S' a metasystem with respect to S, and the creation of S' a metasystem transition. As a result of consecutive metasystem transitions a multilevel structure of control arises, which allows complicated forms of behavior.

We refer to the original system $S$ as the *scope* of the MST, and to the number of integrated systems as its *scale*. The minimal scale of an MST is one. In this case there is no reproduction of the scope, but a control level still emerges, so it is a metasystem transition.

Home
Metasystem Transition Theory
Up
Prev.
Next
Down

MST as the quantum of evolution [empty]

Control hierarchy [empty]

Freedom and

Screenshot from
Principia Cybernetica

Metasystem transitions were introduced in:
The Phenomenon of Science
by Valentin Turchin (1977)

## Levels of control in software

- Machine code - controls elementary operations

## Levels of control in software

- Machine code - controls elementary operations
- Compiler - generates the machine code

## Levels of control in software

- Machine code - controls elementary operations
- Compiler - generates the machine code
- Build system - supervises compilation

## Levels of control in software

- Machine code - controls elementary operations
- Compiler - generates the machine code
- Build system - supervises compilation
- Package manager - orchestrates software builds

## What’s next?

Second cycle with reproducible computations as elementary operations:

- Containers as machine opcodes
- Clusters and grids as hardware
- Disk storage as working memory

## What’s next?

Second cycle with reproducible computations as elementary operations:

- Containers as machine opcodes
- Clusters and grids as hardware
- Disk storage as working memory

- Second-cycle programs: **workflows**

## What’s next?

Second cycle with reproducible computations as elementary operations:

- Containers as machine opcodes
- Clusters and grids as hardware
- Disk storage as working memory

- Second-cycle programs: **workflows**
- As rudimentary today as compilers were in the 1970s
- New challenges for reproducibility