The Software Gardening Almanac

Cultivating Sustainable
Software Development
in the Generative Era

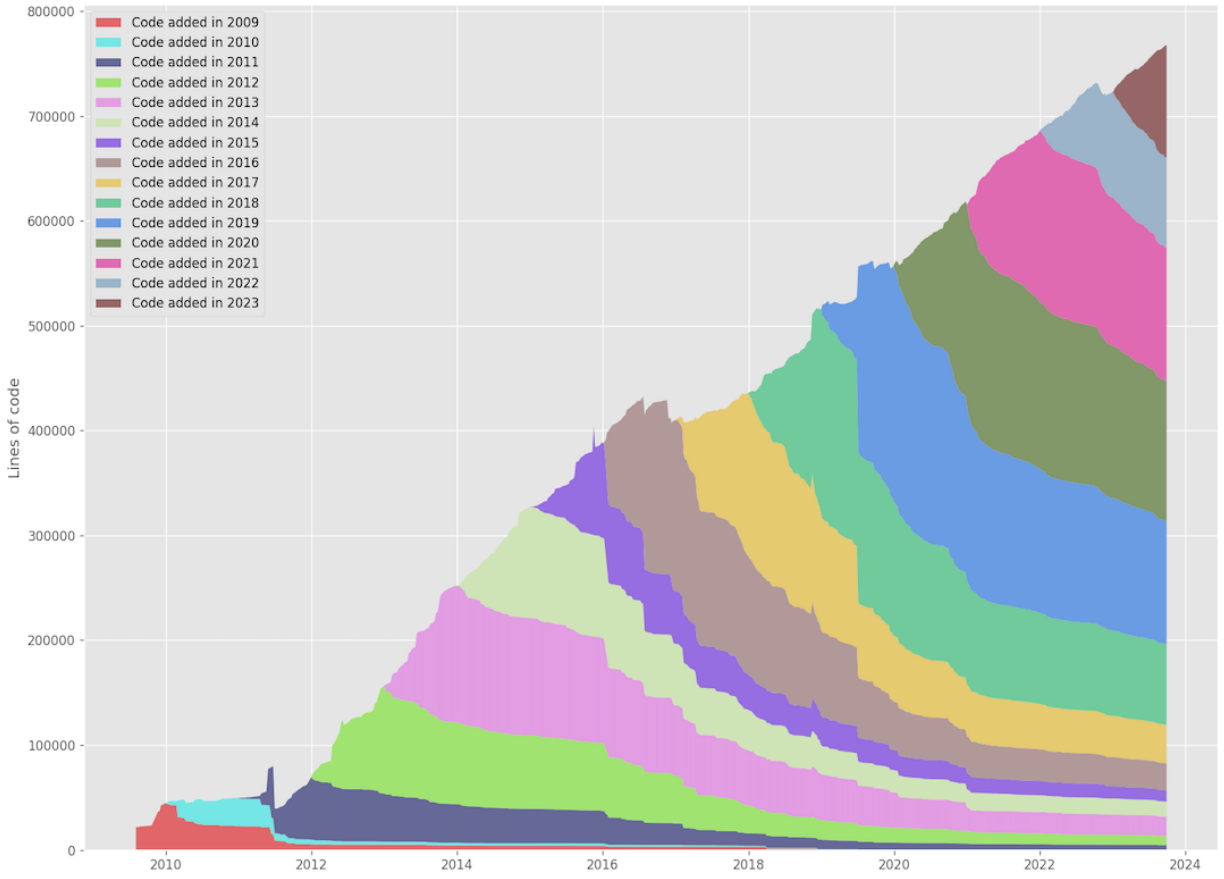
Presentation Outline

1. 🏛️ A philosophical and historical journey
2. 📗 The Software Gardening Almanac
3. 📖 Example sections and chapters

Preface

Preface: expanding and focusing on continuity for ideas found in: Long-Term Software Gardening Strategies for Cultivating Scientific Development Ecosystems

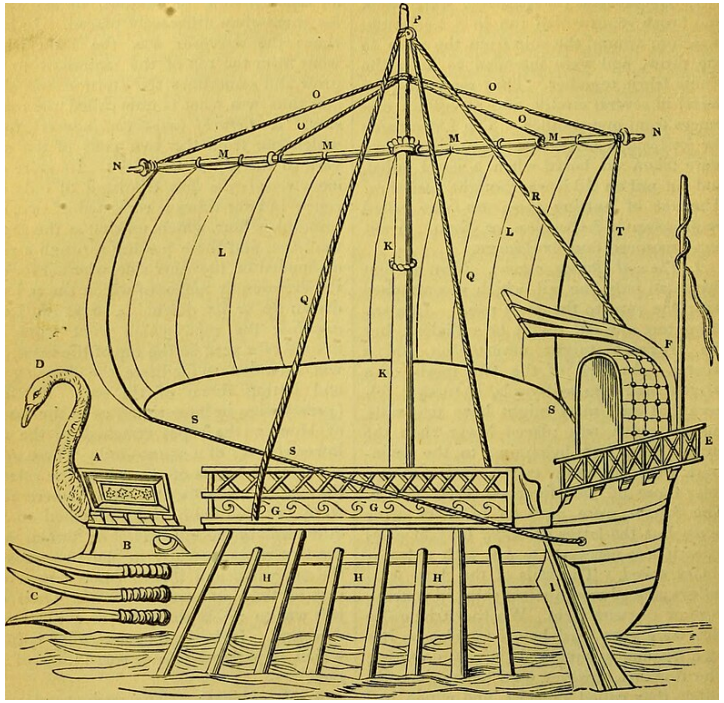
Preface



Python Pandas code additions over time

Some existing investigation in time-based software considerations: git-of-theseus

A philosophical journey

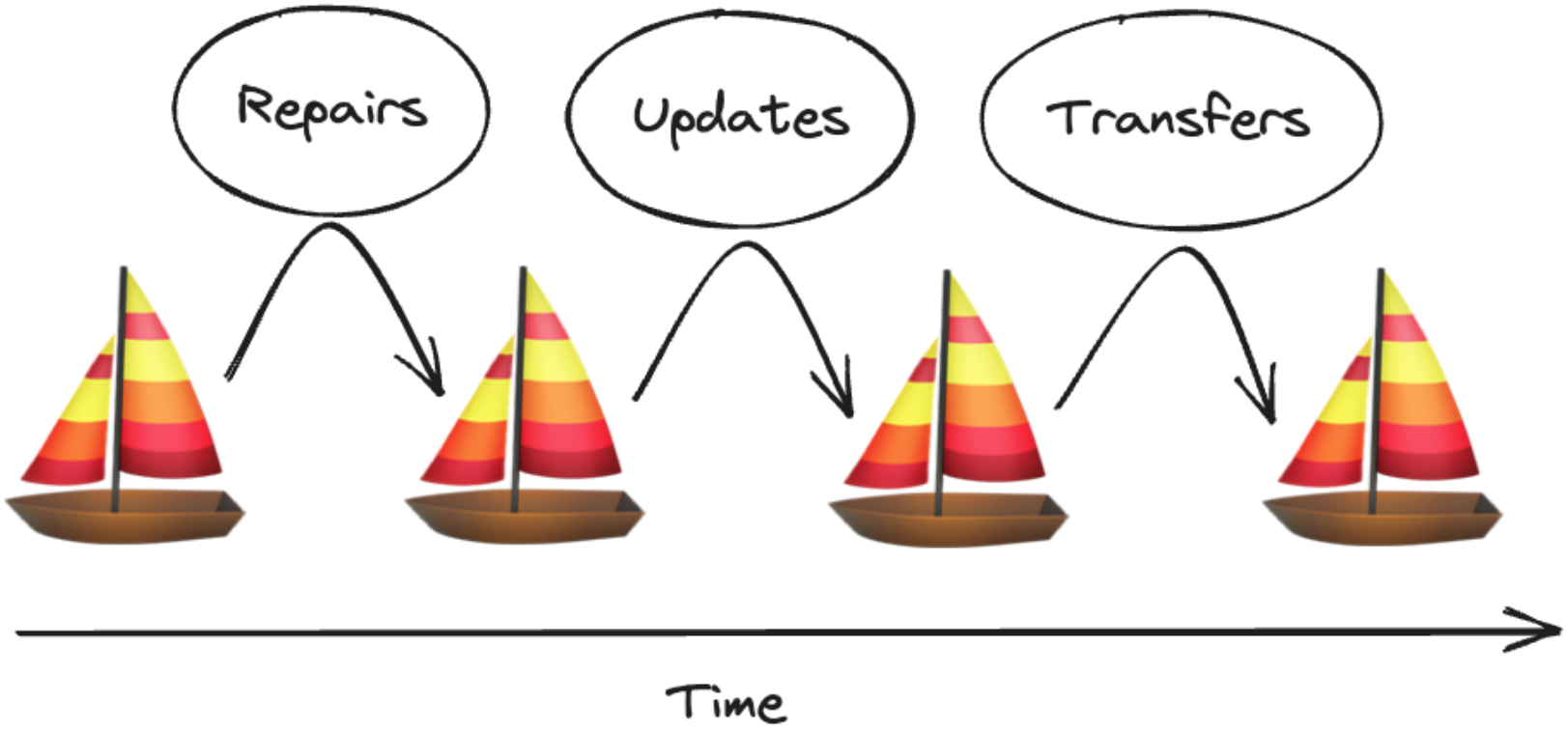


Ship of theseus

"…concerns a debate over whether or not a ship that had all of its components replaced one by one would remain the same ship." (Wikipedia)

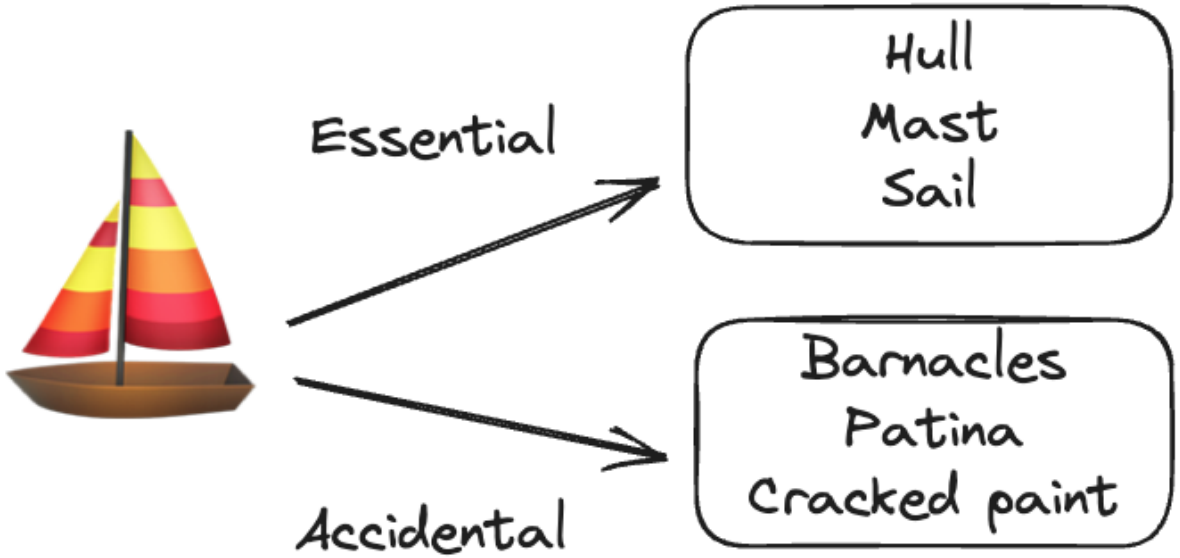
(Image: Smith, William, Sir)

A philosophical journey



- **Continuity vs. Change:** how does the ship's identity continue despite changes which occur?

A philosophical journey



- **Essential vs. Accidental Properties:** Essential properties are those that are necessary for an object's identity, while accidental properties are those that can change without altering identity.

A philosophical journey

How do *“**Software Gardens**”* change over time?

A philosophical journey

"Software Gardens of Theseus"

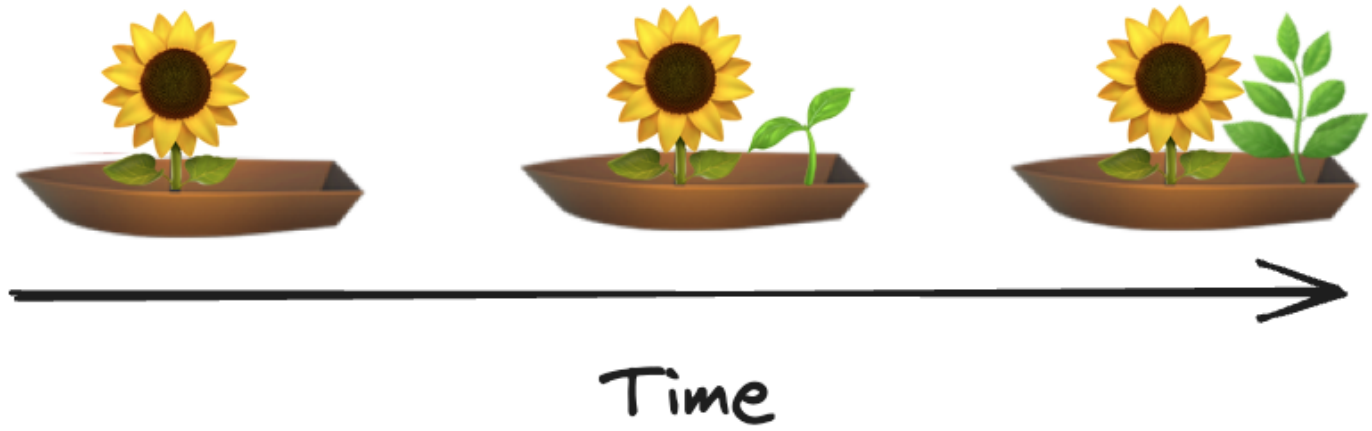
- Gardens on ships? NASA: Lunar, Martian Greenhouses

A philosophical journey

Key aspects for ***“Software Gardens of Theseus”***

- We don’t have many standards of practice for continuity of software.
- We often forget essential properties of our software.
- We often mistake essential properties as accidental.
- Accidental properties might still be important to the garden identity

A philosophical journey



- How does our garden's identity continue alongside *given* change?
- How do essential and accidental properties effect continuity of our software garden?
- A theory: Making our software gardens more essentially and accidentally declarative increases continuity through time.

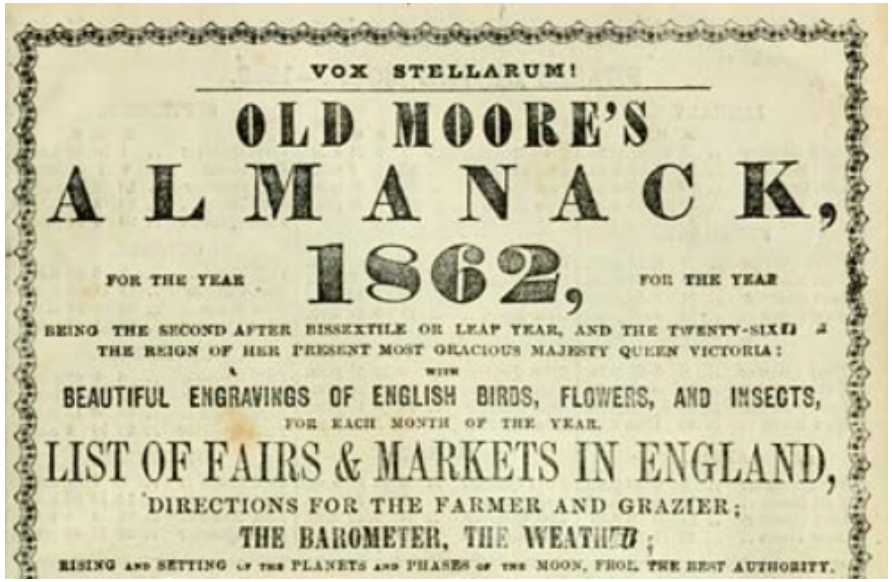
A philosophical journey

- Problem: code is often developed with in a “one-time” waterfall style approach
- The challenge with this is that our code often has a continued lifespan.

Historical exploration

Are there any tools we have historically used to understand and enhance our ability to address change over time?

Historical exploration


VOX STELLARUM!

OLD MOORE'S

ALMANACK,

FOR THE YEAR 1862, FOR THE YEAR

BEING THE SECOND AFTER BISSEXTILE OR LEAP YEAR, AND THE TWENTY-SIXTH OF THE REIGN OF HER PRESENT MOST GRACIOUS MAJESTY QUEEN VICTORIA:

WITH

BEAUTIFUL ENGRAVINGS OF ENGLISH BIRDS, FLOWERS, AND INSECTS,

FOR EACH MONTH OF THE YEAR.

LIST OF FAIRS & MARKETS IN ENGLAND,

DIRECTIONS FOR THE FARMER AND GRAZIER;

THE BAROMETER, THE WEATHER;

RISING AND SETTING OF THE PLANETS AND PHASES OF THE MOON, FROM THE BEST AUTHORITY.

Patterns amidst time and season: **Almanacs**

"[Almanacs include] … information like weather forecasts, farmers' planting dates, tide tables, and other tabular data often arranged according to the calendar." (Wikipedia)

(Image: Old Moore's Almanack from 1862 (cropped))

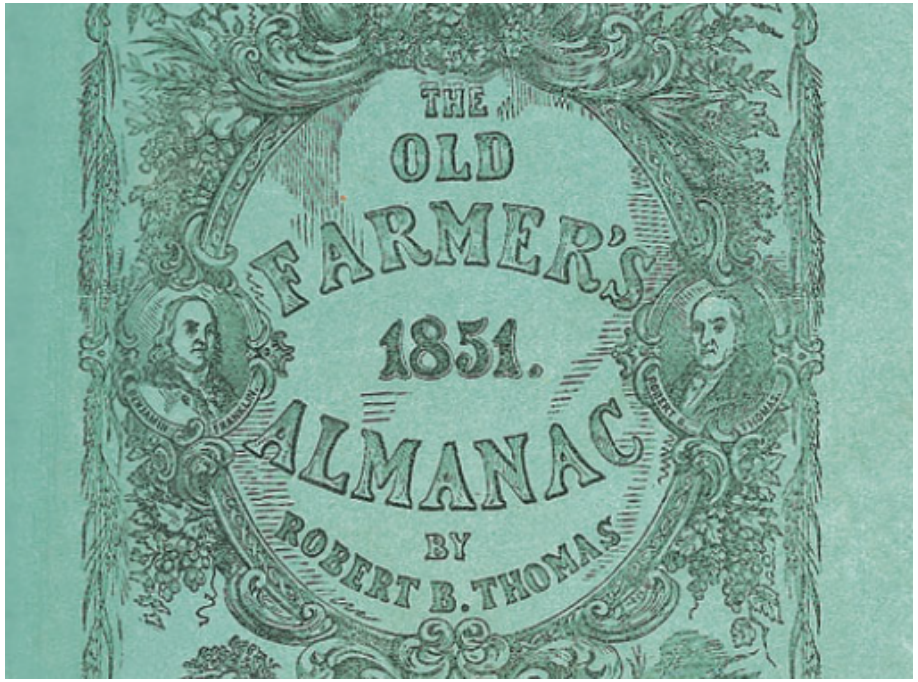
Historical exploration

Menologium Rusticum Colotianum

“A menologium rusticum (pl. menologia rustica), also known by other names, was a publicly displayed month-by-month inscription of the Roman calendar with notes on the farming activities appropriate for each part of the year.” (Wikipedia)

(Image: Accurimbono)

Historical exploration



Old Farmer's Almanac

You can still find almanacs published today like the Old Farmer's Almanac.

(Image: Unknown author)

The Software Gardening Almanac

A living community practicing software gardening to maintain practices

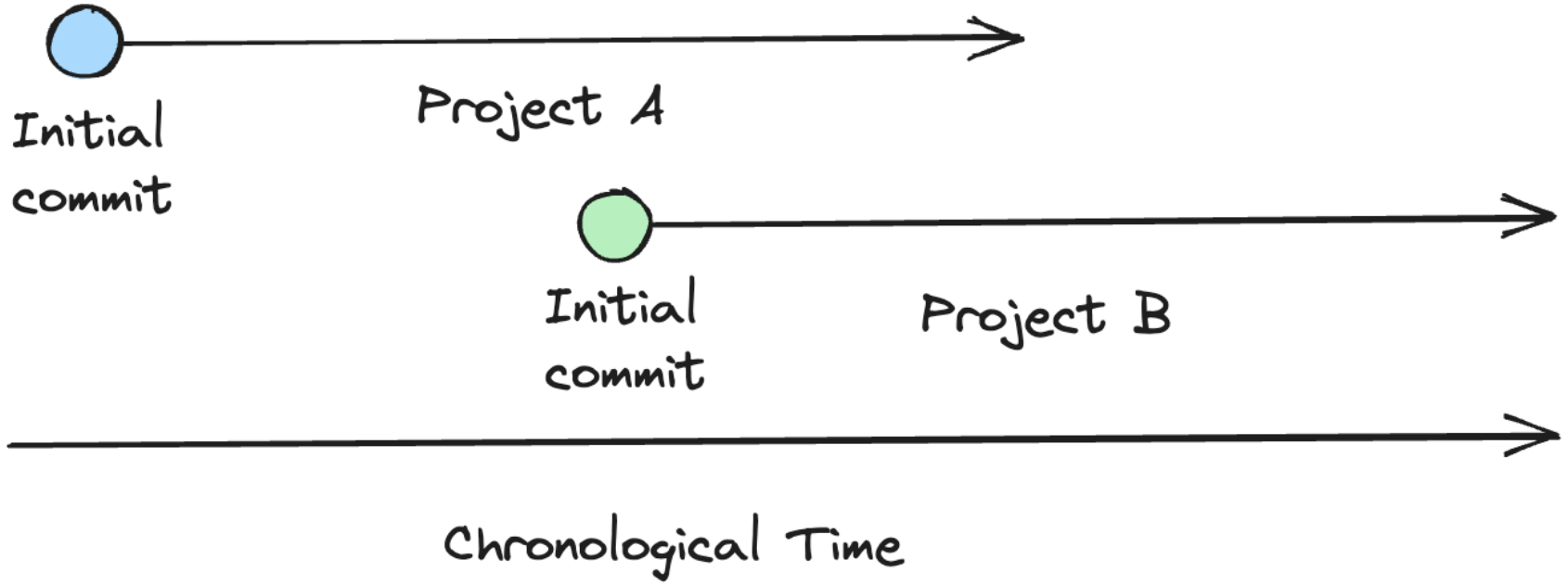
- Planning for scale with common open source, inclusive, collaborative documents
- Providing a referenceable index of codes as “biomarkers” of patterns (like Pylint message index numbers)

The Software Gardening Almanac

Guiding concepts:

- Seeking to define the essential properties of software for highly continuos software.
- Helping the undeclared essential become real for projects
- Shifting the accidental to essential
- Avoiding adventitious accidents

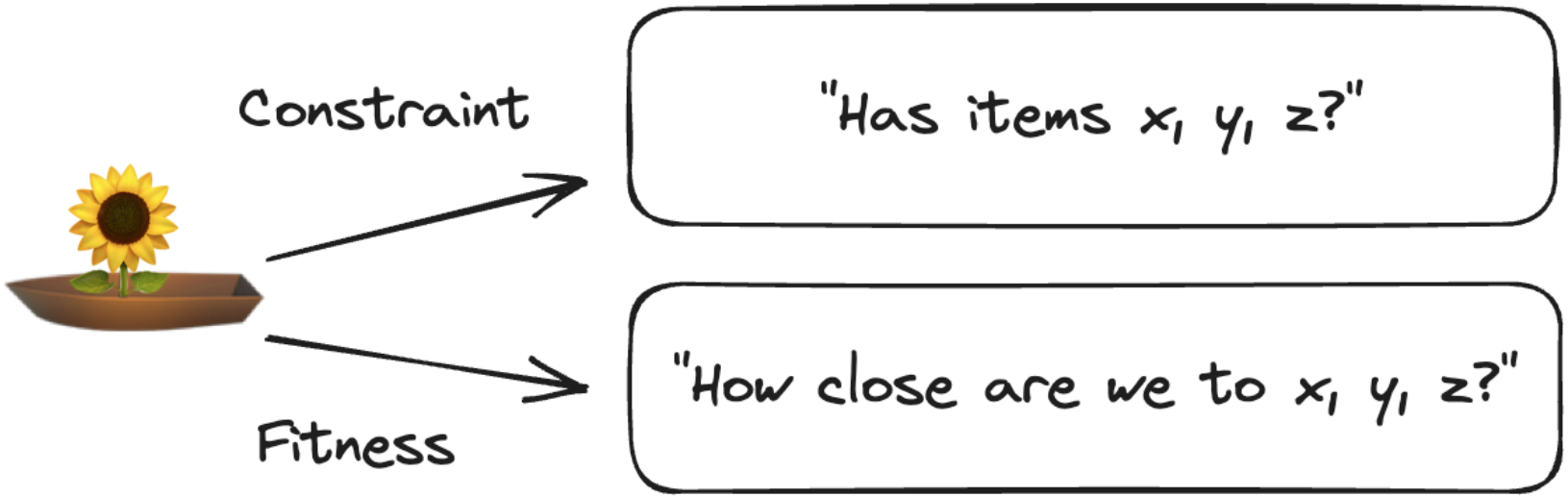
The Software Gardening Almanac



Measurable approaches:

- Relative “project git epoch’s” starting from their initial commit. Using this we can measure the changes in the repository alongside the dynamics of time.

The Software Gardening Almanac



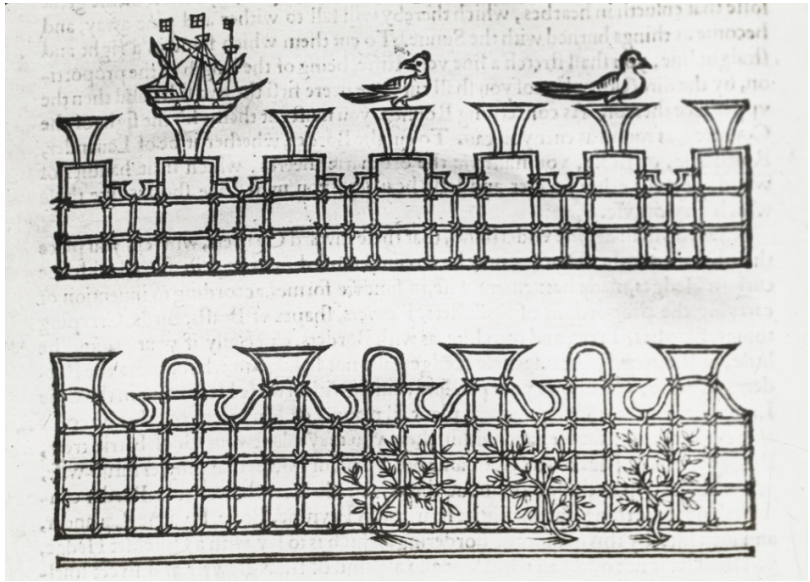
Measurable approaches:

- "Unit-like" constraint testing for precision to expected essential or accidental form.
- Fitness functions for assessing "fit"

The Software Gardening Almanac - parts

- Part 1: 👤 People
- Part 2: 📄 Code
- Part 3: 🕰️ Season

SGA - Part 1 - People

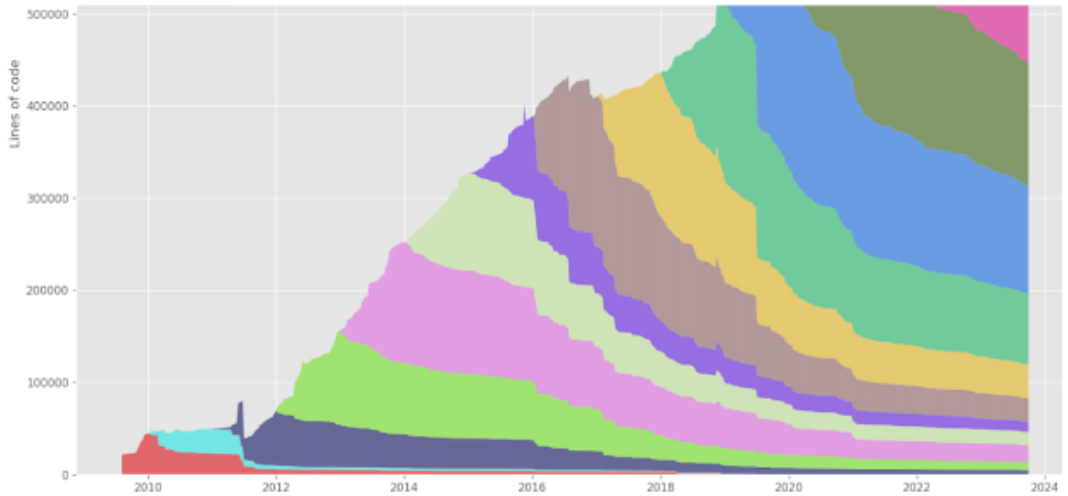


Part 1: People

Software gardener lattices; growing, supporting, and synergizing software gardeners.

(Image: Johnston, Frances Benjamin)

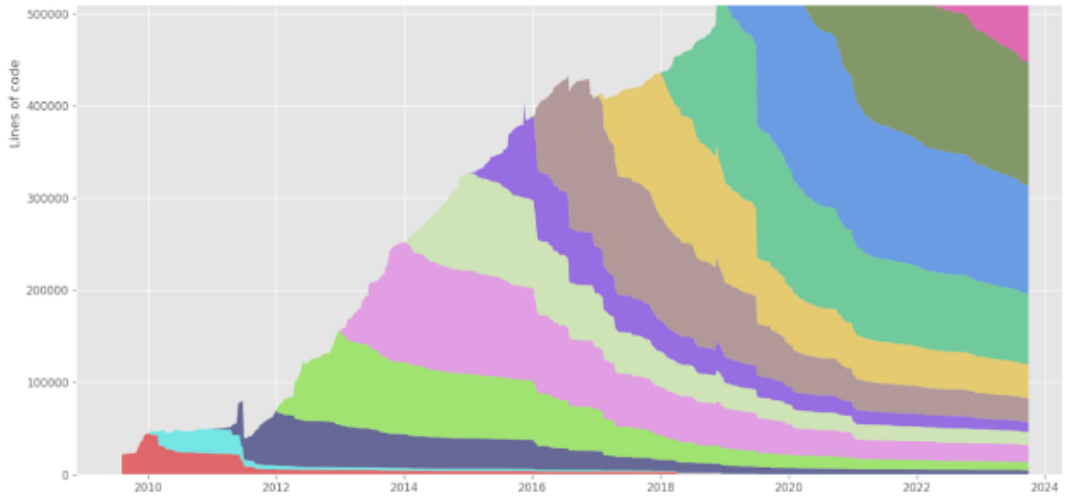
SGA - Part 1 - People



One applied chapter: growing open source; when is it apt to add a new maintainer?

- A lack of contributor diversity can lead to low growth and disaster in the longterm for a project.
- Treating contributors as an essential part of the software process.

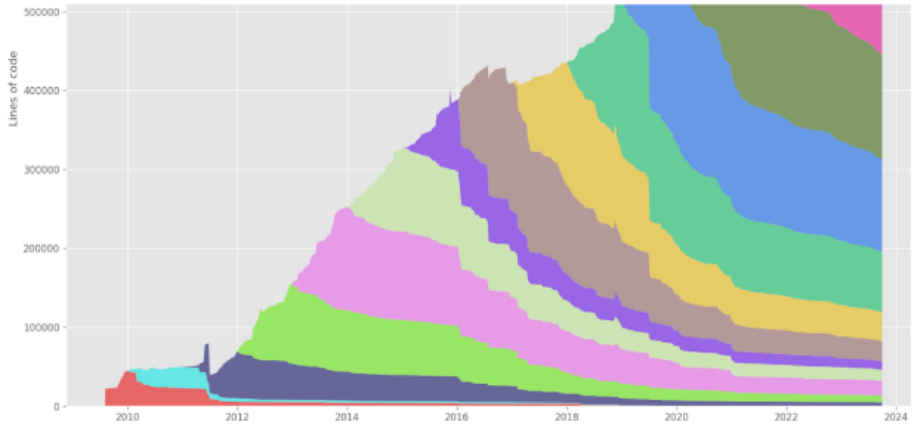
SGA - Part 1 - People



SDA-P0001: **Observation**

- Project has under 1 unique contributor for every X(?) lines of code
- Warning: Grace period of 1 month after initial commit
- Exception: After 1 month

SGA - Part 1 - People



SDA-P0001: **Fix**

- Fix: Introduce the project to more people
- Add a contributors.md document to the repository
- Consider: are there volunteer contributors who have shown interest and would benefit from increased roles?
- Consider keeping the number of contributors >= 3 to retain diversity and resolve “ties”

SGA - Part 1 - People

SDA-P0001: **Additional Resources**

- Wikipedia - Buddy System
- Opensource.guide - Best Practices - Share the workload
- Python Core Developer - Commit privileges
- HBR - Teams Solve Problems Faster When They’re More Cognitively Diverse

SGA - Part 1 - People

More ideas:

- **SDA-P0002:** Repositories with no PR reviews or PR reviews with no reviews / only reviews from the contributor
- **SDA-P0003:** Checking for git “no branching or forking” (commits only to main), and challenges with collaboration
- **SDA-P0004:** Looking at code of conduct inclusion and inclusive / equitable practices

SGA - Part 2 - Code

Part 2: Code

Software gardening senescence; how and why code changes with time.

(Image: Shibata Zeshin)

SGA - Part 2 - Code

One applied chapter: no ecosystem in place for projects

- How can code be healthy if we have no essential properties for the environment?
- Is your software's environment accidental?

SGA - Part 2 - Code

SDA-C0001: **Observation**

- No environment management for project detected
- Detect setup.py, pyproject.toml, renv.lock, environment.yml, etc.
- Unable to infer the project dependencies (as an additional check)

SGA - Part 2 - Code

SDA-C0001: **Fix**

- Add environment management files related to the project language
- Containerization as an optional inclusion?

SGA - Part 2 - Code

SDA-C0001: **Additional Resources**

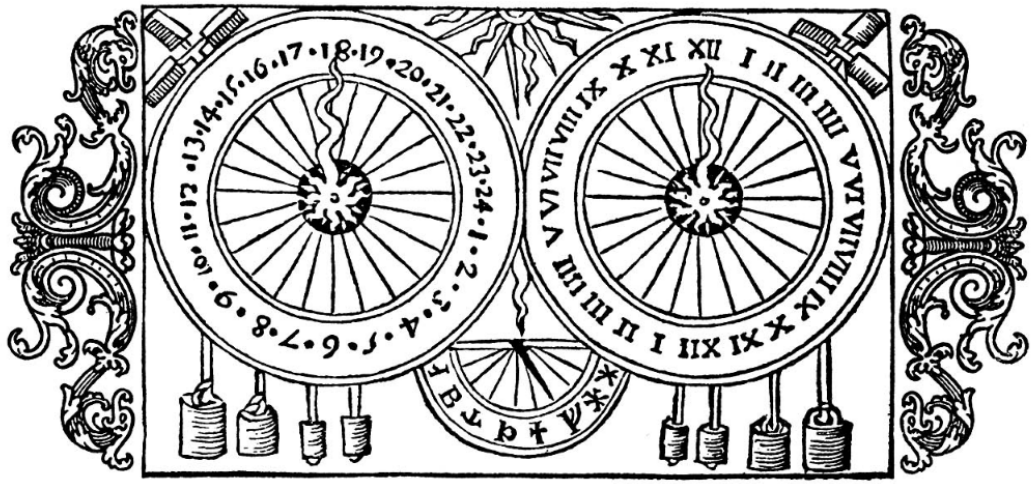
- The Turing Way: Reproducible Environments
- Software Engineering at Google: Dependency Management, In Theory
- Martin Fowler: Infrastructure as code

SGA - Part 2 - Code

More ideas:

- **SDA-C0002:** Detecting repositories which have no testing frameworks installed.
- **SDA-C0003:** Code removal vs code additions over time alongside dead code detection
- **SDA-C0004:** Tight vs loose coupling within the code and how this effects things

SGA - Part 3 - Season

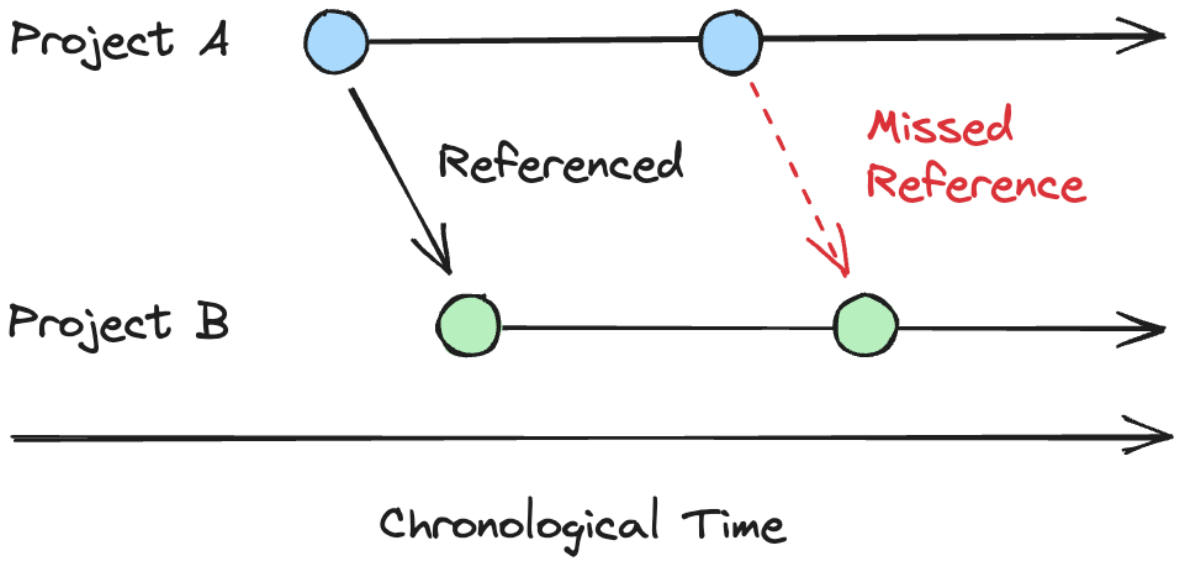


Part 3: Season

Software archeology, nowness, and proactivity; past, present, future, what patterns do we see.

(Image: Olaus Magnus)

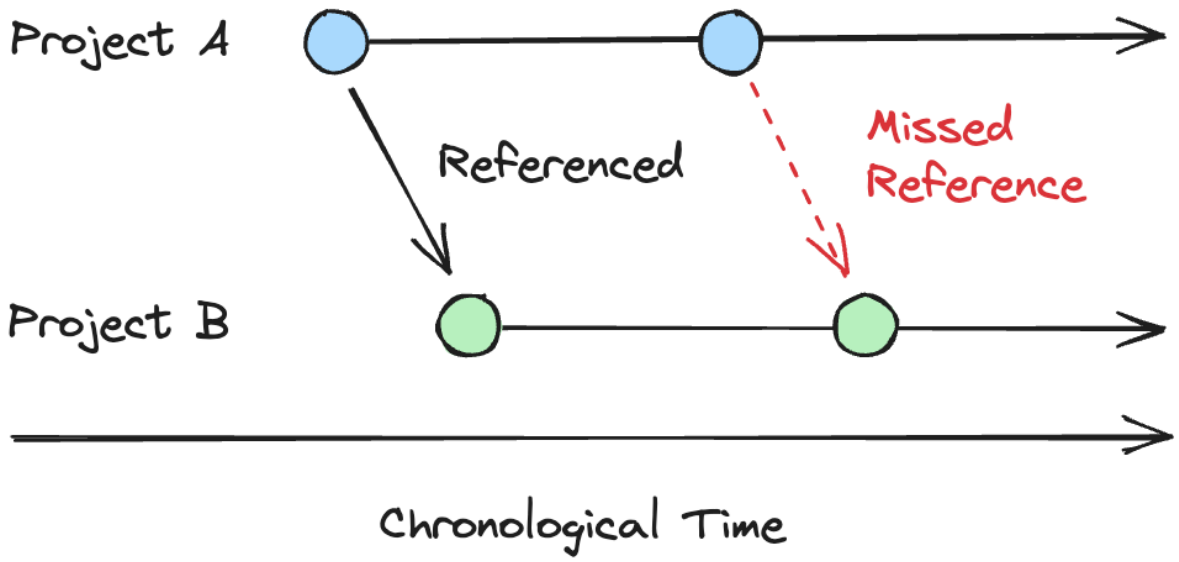
SGA - Part 3 - Season



One applied chapter: ecosystem time synchronization checks

- Disintegrated garden chronology with other timelines
- No automated synchrony checks for repository will lead to accelerated decay

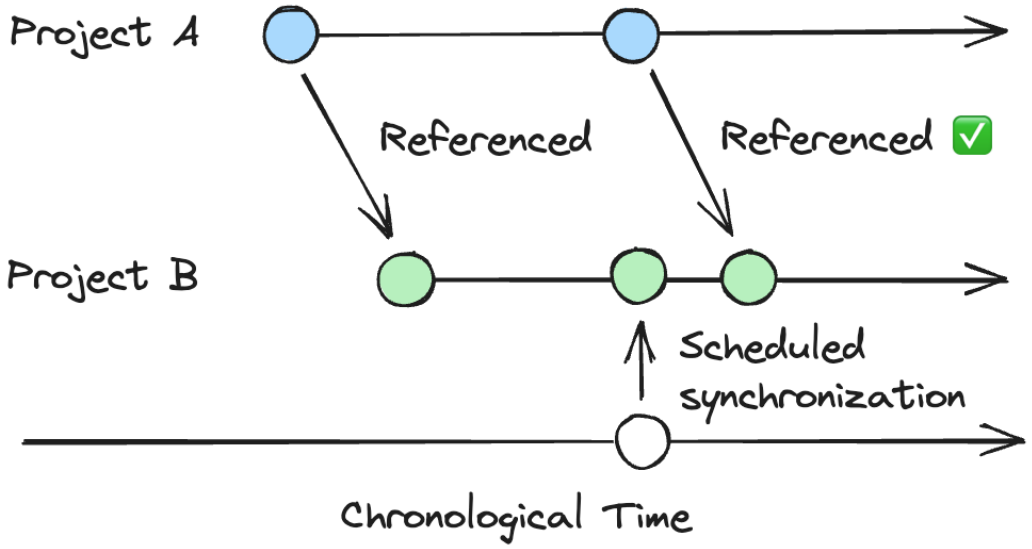
SGA - Part 3 - Season



SDA-S0001: **Observation**

- Detect CI/CD configuration (Jenkins, GitHub Actions, etc)
- Seek chronological triggers which seek common chronological synchrony
- Detect automated checks which occur at a regular rate in order to ensure the project stays up to date with wider environment.

SGA - Part 3 - Season



SDA-S0001: **Fix**

- Add, for example GitHub Actions workflow which runs at chronological schedule (once weekly, for example) to install build of project and run tests.
- Use Dependabot or Renovate to check on dependencies (or more!)
- Use some form of periodic chronological schedule of tests

SGA - Part 3 - Season

SDA-S0001: **Additional Resources**

- Wikipedia - CI/CD (Contiuous Integration, Continuous Delivery +/- deployment)
- GitHub - Working with Dependabot

SGA - Part 3 - Season

Other ideas:

- **SDA-S0002:** Projects where documentation is never updated.
- **SDA-S0003:** Checking on version specification for Python, R, etc.
- **SDA-S0004:** Roadmap and/or architectural decision documentation inclusion to guide projects

Thank you

Thank you! Questions / comments?