

The Scientific Method in the Computational Age

Patrick Ferris

*Department of Computer Science and Technology
Pembroke College*

Supervisor: *Prof. Anil Madhavapeddy*
Cosupervisor: *Prof. Srinivasan Keshav*

The modern scientific method has become computational. But is computer science *helping* or *hindering* research? Using climate science and ecology as a case study, I proposed a systematic study of sources of uncertainty in critical results in these fields. Furthermore, I set out to design and implement a *specification language* and *environment* that empowers climate scientists and ecologists to create less ambiguous, more precise and testable scientific methodologies and results.

June 30, 2024

Contents

1. Introduction	3
1.1. Reading this report	4
2. Progress	5
2.1. Developing Computational, Ecological Methodologies	5
2.2. Systematic Investigation into Uncertainty	6
2.2.1. Uncertainty in Data	6
2.2.2. Uncertainty in Code	6
2.2.3. Uncertainty in Ecology	7
2.3. Incremental Dataflow Pipelines	7
2.4. Reproducible Build Systems	8
2.4.1. Shark	8
2.4.2. Vurl	9
2.5. Specification Languages	9
3. Literature Review	11
3.1. The Specification Language	11
3.1.1. Existing Specifications	11
3.1.2. Typesetting and Formatting	12
3.1.3. Programming Languages	12
3.1.3.1. General Purpose	12
3.1.3.2. Domain Specific	13
3.1.4. Collaboration and Versioning	14
3.2. The Specification Environment	15
3.2.1. Data in the Environment	15
3.2.2. Interactive and Visual	15
3.3. Summary	16
4. Thesis Proposal	17
4.1. Investigating Uncertainty in Climate Science	17
4.2. Design and Implementation of Joe	18
4.3. A Computational Wiki	21
4.4. Timeline	21
A Uncertainty at Scale	23
B Uncertainty in Code using Shark	26
Bibliography	29
Index of Figures	33
Index of Listings	33

1. Introduction

The scientific method is a carefully choreographed dance between domain-specific science and supporting technologies, where the leader and the follower are not always clear. In 1994, Harvard mathematician and public policy specialist Harvey Brooks noted that science contributes to technology by offering “...new knowledge which serves as a direct source of ideas for new technological possibilities” but also that technology feeds into science “...as a source of otherwise unavailable instrumentation and techniques needed to address novel and more difficult scientific questions more efficiently” [22]. In that same year Jeff Bezos incorporated Amazon [28], Tim Berners-Lee founded the World Wide Web Consortium (W3C) [82], and the Linux Kernel reached version 1.0.0 [47].

My work focuses on the interface between scientific endeavour and computers. The insights are derived from a multi-year collaboration with ecologists who are using remote sensing to develop scientific methodologies to assess the health of the planet.

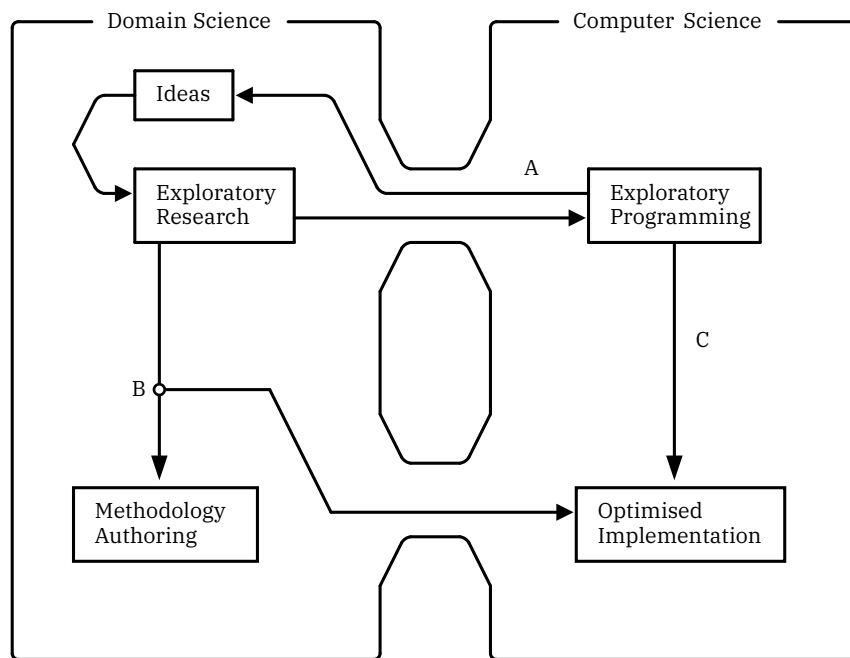


Figure 1: Common pitfalls in the lifecycle of a computational scientific method

Figure 1 shows a simplified lifecycle of a computational, scientific method derived from experiences creating *PACT Tropical Moist Forest Accreditation Methodology* (PACT) [19]. Additionally, it was informed by research into the nature of computer science and software in scientific programming [68, 77]. The working space is divided into the problem domain (e.g. ecology) and computer science with points of crossover and the merging of the two. Three potential points of breakdown are highlighted:

- (A) **Programming influencing ideas and research in the domain:** This is not necessarily problematic, but limitations in research could be misguided by perceived limitations in computer science (e.g. limited resources, lack of programming experience). For example, a scientist may choose coarser resolution raster images due to a lack of computational resource.
- (B) **Fractured methodology writing:** Writing the methodology in most cases takes place in the science’s domain. Little help is offered from tools and software on the computer science side.
- (C) **Error-prone and slow porting of exploratory code to methodology implementation:** Exploratory programming code is often revisited and in some sense made production-ready. At the same time the code is being fed by the research and scientific methodology leading to possible divergence. This is true in the reverse as well – decisions in the code can sometimes alter the methodology in subtle ways and fail to make it upstream to the scientific writing.

Our research shows that these are very real problems impacting our own ecology-based scientific research (Section 2.2). In a world facing a climate and biodiversity crisis [78, 79], the robustness of our scientific research and output has never been more important.

The rest of this report is split into a progress update of the first year (Section 2), a literature view focusing primarily on work related to *specifying* problems in computer science (Section 3) and finally a proposal for the next two years of research (Section 4).

1.1. Reading this report

There are a few extra reading aids scattered across this report. Including:

Question

Questions are moments to engage the reader in directions my mind are going. They could make for discussion points with assessors.

Scenario

Scenarios are design fictions. An idea about what the future could look like if we adopt new practices or technologies. See “*We try to empower them*” - *Exploring Future Technologies to Support Migrant Jobseekers* by Wang et al. for an example of using design fictions [83].

2. Progress

When I hoked there, I would find
An acorn and a rusted bolt.
— from *Terminus* by Seamus Heaney

Pursuing a computer science PhD encompasses much more than just reading around your core topic and writing code. I want to take this opportunity to highlight other ways I’ve *made progress* in the PhD before diving into expected material for this section.

- This year I taught thirteen different students across three colleges for *Foundations of Computer Science*, *Discrete Mathematics* and *Hoare Logic and Model Checking*.
- Since the beginning of 2023, I have presented research for the department at the Alumni Festival, the Cambridge Science Festival, the Cambridge Zero Community Day, the Energy and Environment Group’s research seminar and the Climate and Sustainability Research Showcase.

The work over the past year has been exploratory and varied, ranging from helping develop ecological methodologies (Section 2.1), investigating uncertainty arising from computational aspects of existing research (Section 2.2), building incremental dataflow pipelines (Section 2.3), reproducible build systems (Section 2.4) and exploring specification languages (Section 2.5).

2.1. Developing Computational, Ecological Methodologies

Working closely with ecologists and colleagues in computer science, we developed a precise methodology for analysing avoided deforestation projects in tropical moist forests [19]. For example, here is a short excerpt from the calculation of *additionality* (in this context, the amount of additional carbon stored by an avoided deforestation project).

1. Let T be the set of Pixels in the AOI at time t_0 (**A24, A25**). Let $|T|$ be the number of pixels in T .
2. For each pixel in T find its land use class V .
3. For each year t of time series from t_0 to t_{now} , where t_0 = year of project start and t_{now} is the year of assessment:
 - a. Let $N_{T,V}(t)$ be the number of pixels in each class V in T in year t

Figure 2: An excerpt from PACT Tropical Moist Forest Accreditation Methodology [19]

There are a few design decisions that make the methodology fairly robust and feed into the design of a specification language (Section 4.2).

Explicitly labelling assumptions These appear in bullet 1 of Figure 2 as **A24** and **A25**. These are detailed earlier in the specification. For example, assumption A25 is “The project area is the spatial polygon that defines the area in which the project is primarily operating to conserve forest...”.

Use of mathematical objects and imperative pseudo-code Sets are used throughout along with notions of first-order logic. This is also mixed with a set-based, imperative pseudo-code programming language (e.g. “For each pixel in K_{buffer} and for each potential pixel p in R add p to M if all the following fields match...”).

2.2. Systematic Investigation into Uncertainty

During the interdisciplinary research with colleagues in the [Cambridge Conservation Initiative](#) (CCI), we uncovered sources of uncertainty in our results that were caused by or compounded due to computers. This research was presented as a talk¹ at the [Undone Computer Science](#) conference in Nantes, France this year [32]. The talk abstract has been included in Appendix A.

2.2.1. Uncertainty in Data

Our first source of uncertainty was our input data. One of the most important inputs for the tropical moist forest analysis is the European Commission’s Joint Research Centre’s (JRC) Tropical Moist Forests dataset (henceforth the TMF dataset) [81]. This dataset classifies land in the tropics, at a 30m resolution, into six categories: Undisturbed (■), Degraded (■), Deforested (■), Regrowth (■), Water (■) and Other ().

For example, Figure 3 shows two *versions* of the TMF dataset overlayed onto a region in Indonesia using the same JRC colour schema as above. The data shows land use classes for the year 2008. The image on the left uses the dataset as produced in 2021 and the image on the right uses the dataset one year later in 2022. Large portions of the data have changed to deforested (■). The code for this analysis is [available on Github](#).

There were two main takeaways from this class of uncertainty:

1. **Many datasets that are made available online are not done so in a versioned way.** Furthermore, after download (unless the user does this themselves) the data does not always contain any extra metadata that would help (a) track *data provenance*, (b) allow any other users to detect what version of the data is stored locally and (c) check the data is truly a particular version.
2. **Computations do not track their inputs and label their results.** Even if a dataset was to be properly versioned, it is easy to write scripts and programs that throw this valuable information away making it harder to track *data lineage* over time.

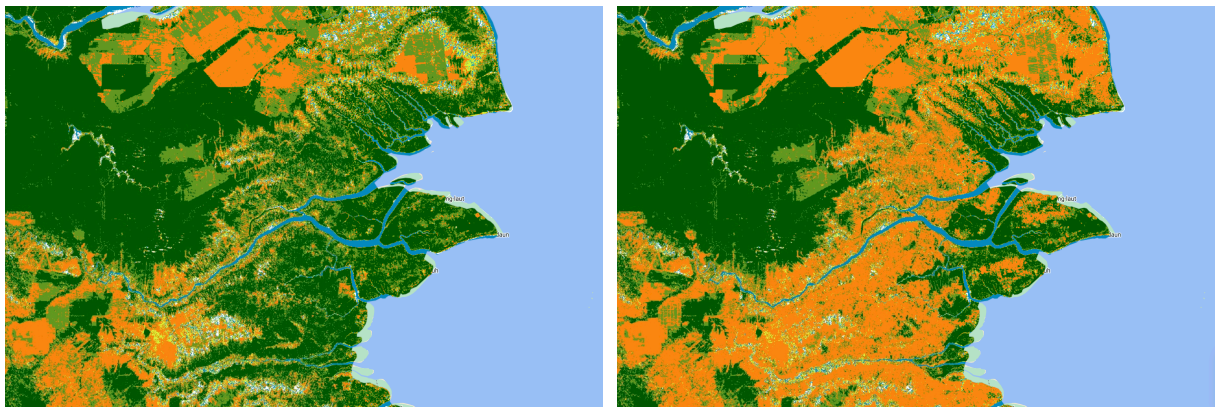


Figure 3: TMF data from 2008 for a region of Indonesia. The image on the left shows the data produced in 2021. The image on the right shows data in 2022.

2.2.2. Uncertainty in Code

Uncertainty in computational, ecological methodologies can arise from the environment in which they execute. This includes the installed libraries and packages on the machine running any analysis.

Between versions 3.2.0 and 3.3.0 of the GDAL geospatial library, the default algorithm for the terrain ruggedness index (TRI) was changed to be one user’s likely wanted². This leads to differences in the resulting TRI map even when the CLI command being run is exactly the same. Appendix B contains a full description of this problem using the reproducible build tool *Shark* (a tool I helped build this year described in Section 2.4).

¹Slides are available: https://patrick.sirref.org/slides/undone_cs_2024.pdf

²It was changed from an algorithm appropriate for underwater terrain to one appropriate for land-based terrain.

```
1 $ gdaldem tri -q /data/elevation.tif /data/tri.tif
```

Listing 1: A simple invocation of the `gdaldem tri` command.

The command remains unchanged but the resulting TRI map is completely different. See the figure in Appendix A for more information and a visualisation of the differences.

2.2.3. Uncertainty in Ecology

The final source of uncertainty explored was in the gap between scientific specification and computer implementation. This arises in multiple interplaying ways:

1. **Computer science is leading ecological science.** Ecological methodologies are becoming more computational and some limitations of the ecological science are not coming from ecology but from what a computer can do as programmed by a *vernacular programmer*³.
2. **A gap exists between how ecologists wish to specify their methods and the code that implements it.** We lack the tools to effectively communicate high-level scientific methodologies in a formal, accessible and easily digestible way.
3. **Computer science enables very complex methodologies.** The ease of integrating more data, more tools, artificial intelligence etc. offered by a computer is both empowering and making it harder to understand the entirety of a single methodology described by one paper. This is compounded by a field still grappling with reproducibility [71].

2.3. Incremental Dataflow Pipelines

After building the implementation for the methodology described in Section 2.1 we needed an underlying engine to execute the scripts, introduce incrementality, improve reproducibility and manage how information flows from component to component. This work reused an OCaml library called `OCurrent`. A value in OCaml of type `'a Current.t` is an incremental, recomputable value of type `'a`.

We leverage a modified version of the `runc`-based `OBuilder` tool in the pipeline. `OBuilder` abstracts two components for performing a more reproducible build: the execution environment (using `runc` on Linux) and storage backend (using ZFS on Linux). Listing 2 is simplified view of how we construct an incremental version of `OBuilder` using `OCurrent`.

```
1 module Input = struct
2   type t = {
3     spec : Obuilder_spec.t;
4     source : [ `No_context | `Git of Git.Commit.t | `Dir of Fpath.t ];
5   }
6 end
7
8 module Output = struct
9   type t = { snapshot : Obuilder.id }
10 end
```

Listing 2: Simplified `Input` and `Output` module for an `OBuilder` plugin to `OCurrent`.

With an extended build specification (the `Obuilder_spec.t`) that allows users to *read-only mount* previous datasets, we control how information flows within the pipeline (e.g. calculating the matching area requires the polygon for the project, so this data is mounted into the matching area container). This is visualised in Figure 4.

³A term coined by Mary Shaw in “Myths and mythconceptions: What does it mean to be a programming language, anyhow?” [74]

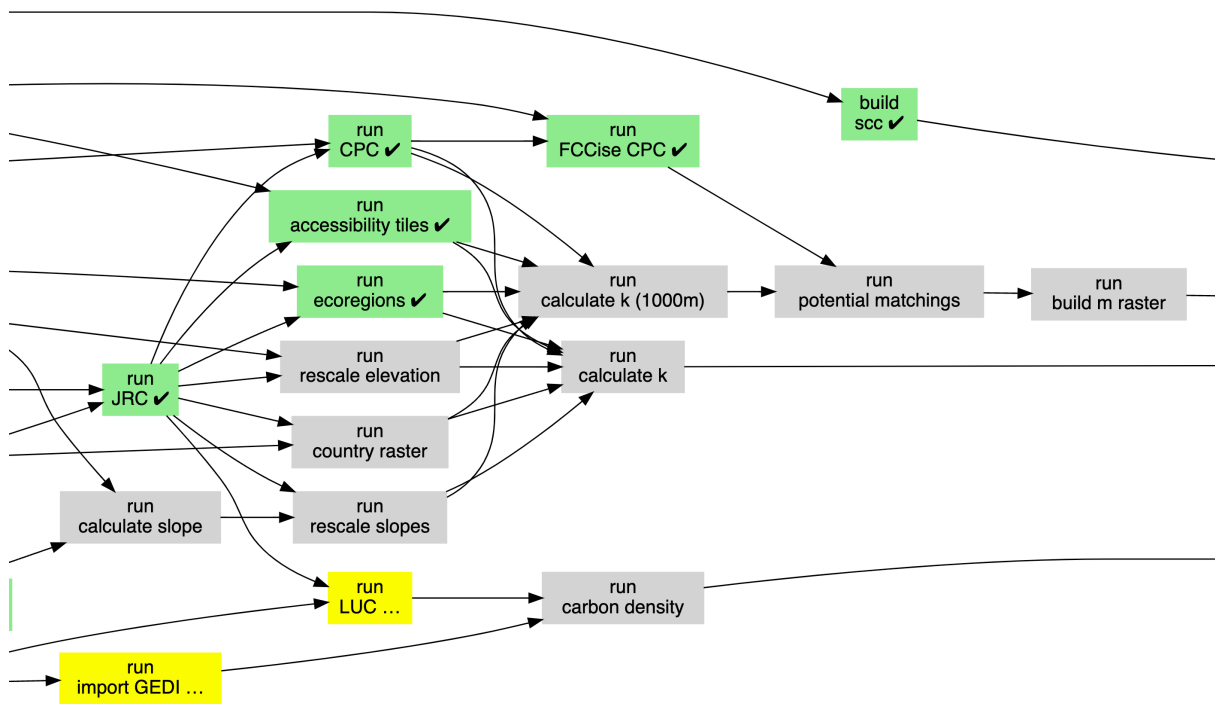


Figure 4: A part of the incremental pipeline for calculating PACT [19], showing how data flows from component to component. Green boxes indicate successfully completed computations, yellow boxes are in the queue to be run and grey are yet to be queued.

? Question

How much incrementality is necessary for pipelines where recomputation is either not that frequent and may also be expensive to perform? This is true in the case where invalidation of nodes in some dataflow graph can over-approximate the need for recomputation. For example, changing a git revision of the code used for a particular build (e.g. by committing changes to the `README.md` – so no impact on the semantics of the underlying program) may cause a cascading, expensive recomputation. We likely need better tools for fine-grained versioning of inputs inside a build system. See Section 3.1.4 for more discussion on this topic.

2.4. Reproducible Build Systems

To bring a robustness to ecological research, we have explored how to systematically remove sources of non-determinism from how results are produced. One example of non-determinism occurs when inputs are not carefully tracked; inputs should also be propagated to results whenever something changes. Another example is removing non-determinism from inherently non-deterministic interactions with the environment, for example the network and name resolution.

2.4.1. Shark

Spurred by the research in Section 2.2 and Section 2.3 we looked for a more succinct way for specifying the interconnected nature of dataflow pipelines. The key insight came when helping colleagues run the implementation described in Section 2.1 which lead to Markdown files with documentation and codeblocks with script invocations.

From this came *Shark* – a tool for specifying bash-like scripts in Markdown that would perform more reproducible builds (see Appendix B for an example of a full Shark program). This reuses the same extensions to OBuilder mentioned in Section 2.3 – execution is sandboxed using

namespacing offered by the Linux kernel and data is stored using ZFS exploiting its snapshotting features, copy-on-write semantics and key-value properties to make the store *self-describing* [56].

2.4.2. Vurl

Vurl (from *Versioned URLs*) was another exploration into reproducibility, this time for the network. More specifically, for name resolution. Vurl would cleverly keep track of how a given name was resolved in order to provide greater reproducibility later when the same name was resolved. A small example shows how a DOI gets resolved to a particular paper (in this case from IEEE).

```
1 let () =
2   let vurl = Vurl.of_uri "https://doi.org/10.1109/SASOW.2012.14" in
3   let vurl, file = Vurl.file vurl in
4   Vurl.pp Format.std_formatter vurl
5
6 {
7   "intentional_uri": "https://doi.org/10.1109/SASOW.2012.14",
8   "segments": [
9     { "uri": "file:./_data/document-6498375",
10       "cid": "bag5qgeraipjyvov4axsmb4pktfhmleqi4oc2lno5if6f6wjyq37w4ktnvcvxq" },
11     { "uri": "https://ieeexplore.ieee.org/document/6498375/",
12       "cid": "bag5qgeraipjyvov4axsmb4pktfhmleqi4oc2lno5if6f6wjyq37w4ktnvcvxq" },
13     { "uri": "http://ieeexplore.ieee.org/document/6498375/",
14       "cid": "bag5qgerap5iaobunfnlovfv4jq2ygp6ltszlrreaskyh3mseky5osh2boq" }
15   ]
16 }
```

Listing 3: An OCaml program using Vurl to resolve a DOI to a file on the local filesystem

Vurl also contains other experimental features like a capability-first API (capability as in *Capability Myths Demolished* [58]) using the Capnproto serialisation protocol.

The research and thoughts on how incremental, reproducible dataflow pipelines fit into the bigger picture of ecological research is further explained in our paper *Planetary computing for data-driven environmental policy-making* [31].

2.5. Specification Languages

Developing a specification of the ecological methodology (Section 2.1) raised many questions about what are the current best methods for producing high-level, formal specifications in other areas of computer science are. I explored various formal methods and theorem proving softwares including Coq and FStar (F★). As part of the investigation into backend storage models for reproducible builds, a common requirement was adding a privacy model. Consider a computation that uses raw data about endangered species – the results should be protected as leaking them may put these endangered species at risk from poachers. I explored the *Decentralised Information Flow Control* model suggested by Myers and Liskov, formalising the approach in F★ and extracting to OCaml [62].

```

1 let anil = User.{ username = "Anil" }
2 let ryan = User.{ username = "Ryan" }
3 let jessica = User.{ username = "Jessica" }
4 let michael = User.{ username = "Michael" }
5 let maddy = User.{ username = "Maddy" }
6 let patrick = User.{ username = "Patrick" }
7
8 let ryans_label = Users.v ryan [ michael; anil ]
9 let jessicas_label = Users.v jessica [ maddy; anil ]
10 let patricks_label = Users.v patrick [ ryan; anil ]
11
12 let v =
13   let* patricks_data = M.return patricks_label "Super " in
14   let* ryans_data = M.return ryans_label " secret " in
15   let+ jessicas_data = M.return jessicas_label "data" in
16   patricks_data ^ ryans_data ^ jessicas_data
17
18 let er = M.label v |> Users.effective_readers
19 (* val er : User.t list = [{User.username = "Anil"}] *)

```

Listing 4: An example program using the extracted OCaml code from $F\star$ to implement a DIFC monad

Listing 4 shows an OCaml program using extracted code from an FStar implementation of DIFC. Labels are propagated using a privacy-preserving algorithm (ensuring result labels are a *restriction* of input labels). Using a general-purpose proof-oriented programming language was both informative for ideas about a specification language (Section 4.2) and potentially useful as a verified component of a specification environment (Section 4.3).

? Question

To what extent is formal verification enough to be confident in the correctness of a program? Formally verifying a piece of code using a theorem prover is a powerful mechanism for adding additional guarantees to your program. Often, however, we wish to *extract* these programs and proofs to another language (or platform) and this can introduce opportunities to undermine the hard work of the theorem prover. For example, when developing the DIFC FStar implementation above, the `FStar_OrdSet` module exposes directly the OCaml type for ordered sets as a `'a list`. A key property that most hold for this list is that it is sorted but we could directly supply an unsorted list and there are no assertions that it is sorted.

3. Literature Review

The gaps are the thing... Go up into the gaps. If you can find them; they shift and vanish too. Stalk the gaps. Squeak into a gap in the soil, turn, and unlock – more than a maple – a universe.

— from *Pilgrim at Tinker's Creek* by Annie Dillard

In Section 1 we proposed a combination of specification authoring systems with tight integration into a system for reproducible scientific programming. As such the literature review is split into considerations for the specification language (Section 3.1) and the specification environment (Section 3.2). There is, of course, considerable crossover between the two.

There are many partial solutions in this space, but none that tackle all the problems we have encountered. This literature review is structured as hike through the landscape of systems, tools and programming languages in the space of scientific programming and specification authoring. As we meander we will collect properties and features that seem fitting for the problem we are solving, before summarising all of this in a gap analysis table (Figure 6).

We start at the “organizational notes for the Internet”: the Internet Engineering Task Force’s *Request for Comments (RFCs)* [1].

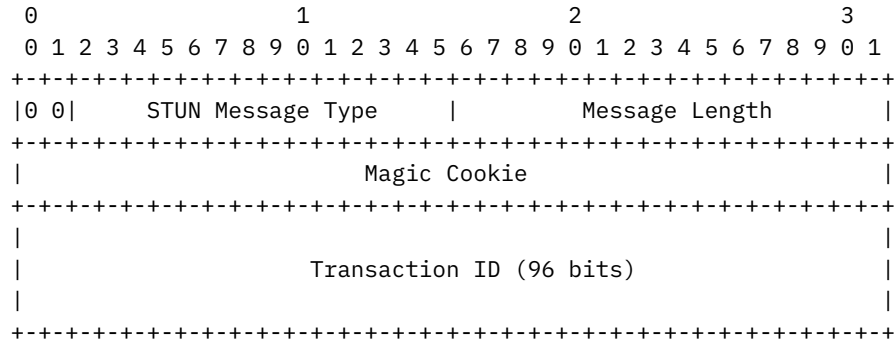
3.1. The Specification Language

3.1.1. Existing Specifications

RFC documents are technical specifications that describe components of the infrastructure of the internet. For example, RFC 9114 describes the third version of the Hypertext Transfer Protocol (HTTP) [20]. The structure of RFCs is also described *in* RFCs (for example RFC 7841 [36]). RFCs exist within an ecosystem of tools (e.g. [xml2rfc](#)) and social infrastructure (e.g. [working groups](#)).

RFCs have been extremely successful. However, their static nature, lack of basic tooling for validation or reasoning about the complex processes they describe and the ambiguous nature of natural language has allowed easily caught errors to sneak through. For example, an error found in the *textual* description of the lower-level interface for the transmission control protocol (TCP) that likely would have been caught with a more structured and validated description of the interface [2]. Furthermore, the specification does not offer *implementations* a means by which to check their validity. There have been several attempts to build tools that improve RFCs and generate code from them [85, 86].

RFCs are also extremely limited in terms of typesetting and formatting. On the one hand, this makes for a commonality across RFCs but also forces authors to use inappropriate, inaccessible communication styles. For example, the *Session Traversal Utilities for NAT (STUN)* messenger header described as ASCII art (Listing 5) [51].



Listing 5: Format of STUN Message Header

One reason for this informality in specification writing is perhaps the low risks associated with implementations going wrong. Indeed, when we look towards software where this is not the case (e.g. avionic control systems) we see more formal approaches to requirement analysis and specification [37, 46]. In the case of *Requirements Specification for Process-Control Systems* in which Leveson et al. discuss a specification language for the Traffic Alert and Collision Avoidance System (TCAS) they recognise the need to bridge a gap between computer scientists and domain users “The example specification demonstrates... the feasibility of building a formal model of a system using a specification language that is readable and reviewable by application experts who are not computer scientists or mathematicians.” [46].

Shifting from informal specifications to formal specifications lands us closer to the world of formal methods and model checking. One of seminal languages in this space is Z [75]. In *Z Formal Specification Language - An Overview*, Ruhela argues that without a formal specification language we are more likely to have *ambiguous, incomplete, vague* and sometimes *contradictory* specifications. For describing concurrent and distributed systems, TLA+ has been favoured for many years [44]. These languages are perhaps too formal for the kinds of specifications we might write in a scientific methodology, however, the tools and theories they are founded on will be useful.

3.1.2. Typesetting and Formatting

Being able to control and format specifications can borrow from the areas of typesetting software. Most computer scientists and mathematicians are familiar with LaTeX, a system for typesetting documents [45]. The macro system introduced in TeX and LaTeX is often a stumbling block for many. Typst⁴, a new typesetting software, overcomes this problem by using more familiar functional-style programming constructs to empower users to control how their documents are rendered [3]. Simpler, lightweight markup languages like Markdown are also potential ways to write specifications but could lack the necessary structure and flexibility [38].

3.1.3. Programming Languages

3.1.3.1. General Purpose

Often, programming languages are used to *solve* problems rather than *specify* them. However, there is a wealth of knowledge to be gained from the design and implementation of programming languages in order to make them less error-prone, easier to debug and more pleasant to write.

Python and R are very common programming languages used in ecology (and scientific computing more generally) [43, 76, 84]. It is important to take into consideration the familiarity data scientists and ecologists have with these languages and their paradigms (in particular an imperative, object-oriented approach). There is also great familiarity with *declarative* languages such as SQL which may or may not be embedded within a general-purpose language via a library [18]. Being declarative (i.e. stating the intent of the program not the control) seems fitting for specifications [53].

⁴This document was written using Typst.

Continuing along this spectrum we arrive at functional languages whose core features are gaining in popularity [24]. New ideas from this field include Hazel: a *live* functional programming *environment* organised around *typed holes* [66]. With typed holes, the Hazel environment can partially evaluate and typecheck programs that are syntactically incomplete. Figure 5 shows a simple projection of the first and second elements of a pair but where we have not completed the value of `v` yet. Hazel tells the user both the partially evaluated answer *and* the expected type of hole in the program.

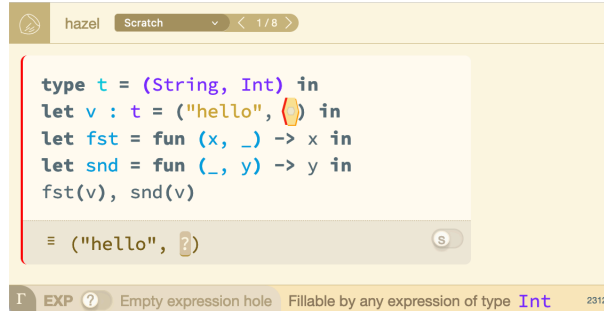


Figure 5: An example of a Hazel program evaluation even when it is incomplete

Hazel has spawned additional user-oriented research with a theoretic grounding including total type error localisation and recovery [87], contextualised programming language documentation [70] and direct integration of graphical user-interfaces into the language environment [64].

3.1.3.2. Domain Specific

Branching from our general-purpose programming language path are various domain-specific languages (DSLs) that offer insight into what is possible when relieved of the burden of being flexible enough to solve a large diversity of problems. Before considering DSLs themselves, it is worthwhile discussing *Racket* the language-oriented programming language descended from Scheme [4]. In the “Racket Manifesto” Felleisen et al. describe the power of the language-oriented approach.

An abstraction enforces invariants. Languages are abstractions, and their creators must have the means to build the necessary enforcement mechanisms – especially when components in these languages end up in an interconnected, multi-lingual contexts. Since Racket is a language for building programming languages, it supplies the building blocks for the construction of enforcement mechanisms, too.

The multi-lingual context, particularly for specification writing is very attractive (recall in Section 3.1.1 the use of ASCII art, state diagrams alongside prose etc.). The Pollen markup language is completely written in Racket [5]. Listing 6 contains a small example of pollen markup – an example of a DSL for flexible typesetting.

```

1 #lang pollen
2
3 ◇headline{Hello from Pollen}
4
5 ◇items{
6   ◇item{We're still in ◇link["https://racket-lang.org"]{Racket!}}
7   ◇item{Pollen is a syntax for flexible markup}
8 }
```

Listing 6: A snippet of Pollen markup

DSLs are a promising avenue for enabling targeted solutions that incorporate domain knowledge into their design helping adoption and their efficacy [55]. A good example of this is Catala: a

domain-specific programming language designed for writing literate, correct legislative texts [54]. An example is given in Listing 7 showing a partial specification of the US tax code.

```
1 The term "qualified employee discount" means any employee discount with
  respect to qualified property or services to the extent such discount
  does not exceed
2
3 (A) in the case of property, the gross profit percentage of the price at which the
  property is being offered by the employer to customers
4
5 scope QualifiedEmployeeDiscount :
6   definition qualified_employee_discount
7     under condition is_property consequence
8   equals
9     if employee_discount >=
10      customer_price * gross_profit_percentage
11    then customer_price * gross_profit_percentage
12    else employee_discount
```

Listing 7: An example of Catala describing a section of the US Tax Code [54]

There are multiple desirable properties to be taken from Catala:

Literate programming style Coined by Donald Knuth, literate programming is a method of combining a programming language with a documentation language: “Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want the computer to do” [42].

Domain-specific Syntax Working closely with legal experts the language has evolved to include quality-of-life improvements that meet their needs (e.g. syntactic sugar for expressing percentages).

Formalisation using F★ Different internal parts of Catala have been formalised using the F★ proof-oriented programming language. In turn, this has created reusable components like a date-time library with a formal semantics [59, 60].

Similar properties also exist in the world of specification languages. Take for instance *LEM* a research specification language that aims to free people from “proof-assistant lock-in” and provide a “language of executable mathematics” with “human-readable output” for the “working specifier” [61]. It is possible to make some compromises to the expressivity of the safety guarantees and look at specification languages that use *testing* as their means of checking whether an implementation is correct (e.g. property-based testing [21]).

3.1.4. Collaboration and Versioning

Writing scientific methodologies and their specifications does not happen in isolation. Experience working with ecologists at the CCI suggests there is a need for better models and tools to enable collaboration and versioning.

In terms of collaboration there has been a recent surge in interest in data-structures and methods that support a notion of local-first development [41]. Whilst local-first software improves data sovereignty, it also introduces mechanisms (e.g. Conflict-free Replicated Data Types) for easing collaboration [48]. The space of CRDTs is large with different trade-offs [17]. However, the approach is also amenable to version control [39, 50]. CRDTs do seem to be a good fit for scientific methodology writing which is perhaps a slow, offline process. Handling version control automatically, or at the very least, in a way that guides a user reduces the need to re-train scientists to use tools such as Git. It is noted in a biology paper assessing the usefulness of Git that “While time invested in becoming proficient in Git would be valuable in the long-term, most scientists do not have the luxury of learning software skills that do not address more immediate problems.” [72].

Versioning is not restricted to how we collaborate on a single document but also how we compose across documents and codebases. In software this has led to package management. Some programming languages have made this problem a part of the language itself rather than

an ecosystem problem. For example, definitions within the Unison language are identified by the *hash of its syntax tree* (content-addressed) and are persisted as the format of the language itself [80]. This does mean there are limitations when working with tools that expect a plain-text view on files (e.g. Github).

3.2. The Specification Environment

The focus thus far has been on lower-level details of specifying scientific methodologies. We turn now to a higher-level consideration which is the context in which these specifications are written and interacted with. Specifications should be deterministic without impeding how easy they are to write. This is particularly acute in the scientific methodologies where *datasets* are interacted with directly introducing two sources of additional uncertainty: the network and the filesystem. Data science is often dynamic and visual [16]. The environment should also reflect this property of data science.

3.2.1. Data in the Environment

Datasets are diverse: in size, in format, in how valuable they are. When ingesting data into a specification we need to take into consideration how to reduce the possibility for non-determinism. Versioning data from possibly non-deterministic sources is not new for example the import system in the `go` programming language [73] and docker data management using images with unique hashes. In academia there have been attempts to integrate versioning into the environment to help with non-linear, exploratory programming [40]. Other tools have been suggested that can take existing workflows and provide a reproducible package of code and data using system call analysis and provenance [25]. There is a tension between post hoc and ante hoc reproducibility; the former likely offers more flexibility to users to explore whilst the latter will likely capture more information.

? Question

In a specification language, to what extent do we need privacy tools? In the context of an implementation of a methodology ensuring the confidentiality and integrity of data is important. However, does this translate to a specification of that methodology. Possibly depending on what the specification is allowed access too. In such a system we may need to look towards theories like differential privacy [29] and decentralised information flow control (DIFC) [62]; working them directly into the environment.

Handling data in the environment would also benefit from *describing* the data. Our evaluation domain, ecology, is particularly subtle in this regard due to various kinds of datasets we see including: raster images (e.g. GeoTIFF [63]), vector formats (e.g. GeoJSON [23], TopoJSON [57], WKT [67] etc.), geospatial columnar format (e.g. geoparquet [6]) etc. We could be interested in describing these formats in the same vein as something like Packet Types [52] (at the bit level), but perhaps more usefully a high-level description of the data would be better. We see this in data description formats like JSON Schema [69] and cross-language tools like *adaptable type definitions* (ATD) [14]. This high-level approach to *typing* data also plays well with type systems, like gradual typing [49]. We have also discussed the benefits of the declarative SQL query language which is founded on Codd's seminal work on data description via a relational model [27].

3.2.2. Interactive and Visual

The specification environment should reflect the way in which domain experts explore, debug and think about their science. In ecology, this is often visual and explored through tools like QGIS [7] and integrated development environments (IDEs) like RStudio [8]. Earlier (Section 3.1.3.1) we mentioned how the Hazel programming environment is also looking to explore GUI-based programming experiences to help better support common scientific workflows [65]. A relatively new, all-in-one platform for data analysis, visualisation and documentation is Observable [9]. It's

integration of programming language experience (primarily in JavaScript) and execution environment is very close to the vision for the specification language and environment.

3.3. Summary

We have visited many programming languages, specification authoring tools, coding environments and version control systems. They are summarised in the following table as they relate to the various properties that have been discussed and would make a good specification language and environment. It should be noted that for brevity the exact reasoning for each decision has not been given. Where a question mark appears (?) it could be argued either way. For example, is the Unison programming language *collaborative* – with features like built-in version control and a platform for distributed computing it could be argued it is geared more towards collaboration than say Python.

Kind	Description	Usability			Expressiveness			Fit for purpose	
		Graphical	Literate	Familiar Syntax	General Purpose	Declarative	Executable	Versioning	Collaborative
Programming Languages	Python [15]	x	x	✓	✓	x	✓	x	x
	R [10]	x	x	✓	✓	x	✓	x	x
	Racket [4]	x	x	x	✓	x	✓	x	x
	Matlab [11]	x	x	✓	✓	x	✓	x	x
	Hazel [66]	✓	x	x	✓	x	✓	x	x
	Unison [12]	x	x	x	✓	x	✓	✓	?
Specification "Languages"	Z [75]	x	x	?	✓	✓	✓	x	x
	TLA+ [44]	?	x	x	x	✓	✓	x	x
Data Description Languages	PacketTypes [52]	x	x	x	x	-	-	x	x
	JSONSchema [69]	x	x	✓	x	-	-	x	x
	ATD [14]	x	x	?	x	-	-	?	x
DSLs	Catala [54]	x	✓	✓	x	✓	✓	x	x
	SQL [35]	x	x	✓	✓	✓	✓	x	x
	Datalog	x	x	x	✓	✓	✓	x	x
Typesetting	LaTeX [54]	x	✓	x	✓	✓	✓	x	x
	Typst [35]	x	✓	?	✓	✓	✓	x	✓
	Pollen [5]	x	✓	?	✓	✓	✓	x	x
Frameworks	Observable [13]	✓	✓	✓	✓	x	✓	?	✓
	RFCs [33]	x	?	✓	?	-	x	?	?

Figure 6: Gap analysis table

4. Thesis Proposal

Never before had I believed or suspected that I had a soul but just then I knew I had. I knew also that my soul was friendly... For convenience I called him Joe. I felt a little reassured to know that I was not altogether alone. Joe was helping me.

— from *The Third Policeman* by Flann O'Brien

The central question of the proposed thesis is:

Does computer science help or hinder science?

Already there are strong indications that the answer is both (Section 2.2), and so a revised question might include more cautious phrasing like “to what extent...”. More concretely then, we might ask:

How can we improve the interface between computer science and data-oriented science to help express scientific methodologies and improve reproducibility?

I posit that with better tools that focus on writing reproducible scientific methodologies we can help bridge this gap and produce better, more accurate science in the process. There are critical ways in which computer science is hindering important scientific work. My thesis seeks to explain why this is the case using ecology as a case study and offer specific and targeted solutions to these problems (Section 4.2, Section 4.3)

These questions I believe are worth researching given the prevalence of this issue in modern, scientific research and more specifically what is at stake in the domain of climate science and ecology. Whilst there have been many programming languages, specification languages and interactive environments designed in the past, I believe that building something that fits this niche and has the properties described in the gap analysis (Figure 6) is novel.

A successful thesis is one that demonstrates that many of the issues we faced in the creation of the PACT methodology could have been avoided using Joe. In addition to this, I would like to conduct user studies to assess how well the proposed specification language empowers ecologists. The thesis can then show how the various contributions help mitigate broader issues found in the opening chapter investigating uncertainty more widely in climate science and ecology (Section 4.1).

4.1. Investigating Uncertainty in Climate Science

Building on the work this year presented at the *Undone Science Conference* (Section 2.2) I would like to more formally investigate the extent to which computer science (be it data management, methodological complexity, unreproducible environments etc.) impacts results in climate science. This will form a foundation upon which to build and evaluate a specification and documentation language for scientific methodologies (Section 4.2) and the infrastructure and operating system inside which such a language can exist (Section 2.4).

There are prime examples from our own research groups and the CCI to explore these uncertainties, for example in the *Tropical Moist Forest Accreditation Methodology* and *LIFE: A metric for quantitatively mapping the impact of land-cover change on global extinctions* [19, 30].

The goal of this stage of research is not to shame other researchers or entire fields, but to systematically understand the factors that lead to uncertainty in final results, particular as it pertains to the gap between specification and implementation.

Concretely, I see this research taking shape as an experience report in reproducing results from a selection of climate science papers and details of how results could vary given any ambiguities in inputs, methods, libraries etc.

4.2. Design and Implementation of Joe

Scenario

Alice is an ecologist. The methodology she has in mind is rather simple: using the existing tropical moist forest methodology she wants to use a new matching variable, soil type, to improve the quality of counterfactual pixel matches. Using Joe she generates a data description of the new data from an example of the dataset she will use. Alice reviews the uploaded example dataset on a map generated by Joe. Joe warns Alice that unlike the other datasets this one is at a 250m resolution and will need to be upsampled. Alice annotates the best way to do this using an upsampling method already used in the existing methodology. She recompiles. Joe asks if she would like to promote new types to the columnar data for counterfactual pairs now that there is a new matching field and warns that the linked example data no longer typechecks. Alice patches the example data with her own data and Joe ensures the standardised mean difference tests still hold. Alice is happy with her changes and reviews the patches graphically making detailed notes about the changes. Alice uploads the patched methodology to her namespace and shares a link with a colleague.

I propose to implement joe, a functional specification language for describing scientific methodologies. To give a taste of joe, the following example is a partial rewrite of a simplified section of the *PACT: Tropical Moist Forest Accreditation Methodology* [19] (see Section 2.1).

Joe has the following design considerations:

For vernacular programmers Joe first and foremost is targeted at vernacular programmers. Joe is not a programming language in and of itself, but a means and a tool to help communicate computational methodologies from a domain expert to programmers who can implement those ideas.

```
1  ## Matching Variables
2
3  There are several important matching variables to consider when deciding if two
4  geospatially disparate pixels are similar.
5
6  ~~~joe
7  matching-variables = [
8    "elevation",
9    "slope",
10   "accessibility"
11 ]
12 ~~~
13 ...
14
15 ## Good Counterfactual Pairs
16
17 After calculating a set of pairs of matching pixels using the @matching
18 methodology, we develop a notion of a "good" matching. In particular, for each
19 @matching-variables we calculate the standardised mean difference.
20
21 ~~~joe
22 input matching = @matching-variables
23 input pairs : Data.Column
24
25 forall m in @matching-variables in {
26   matched = select $m from pairs
27   forall v in matched {
28     Stats.std_mean_diff(matched). v < 0.2
29   }
30 }
```

Listing 8: An example of a geospatial specification written with Joe

Describing data with types and predicates When writing specifications users don't want to spend a lot of time on typing and specifying data formats. A lot of the time, these types could be inferred by how you access data, reading files on disk, downloading data from the internet etc. Joe keeps track of all of this when you run your program and then offers a more precise version of your specification once you have finished. This could be seen as a form of gradual typing. Consider the example in Listing 8. Initially we wrote that the pairs are some form of generic `Data.Column` format. We can infer more about the type of this data by how it is used. Through a combination of the universal quantifier over the set of matching variables and the relational query used to select rows from the pairs we know that `pairs` must contain columns for "elevation", "slope" and "accessibility". Joe can promote this information back into the types.

```

1  ~~~joe
2  input matching = @matching-variables
3  -input pairs : Data.Column
4  +input pairs : Data.Column(columns=matching-variables)
5
6  forall m in @matching-variables in {
7    matched = select $m from pairs

```

Listing 9: Joe suggesting to promote back better type information

Example-Driven Type Refinement As eluded too in Section 4.1, Joe will likely sit in the context of a reproducible, deterministic environment and can leverage these properties to ease the process of specification authoring. Joe can derive type information through user provided examples.

```

1  We require the JRC Land use class dataset in order to calculate carbon
   densities for tropical moist forests.
2
3  ~~~joe
4  input jrc_raster : Geo.Raster
5  ...
6  ~~~
7
8  ~~~joe-example
9  example jrc_raster =
10     "/examples/JRC_TMF_AnnualChange_v1_1990_AFR_ID16_S10_E10.tif"
11  ~~~

```

Listing 10: An example of a geospatial specification written with Joe

In Listing 10 the user is describing the TMF dataset and has provided an example tile from that dataset. Joe comes with built-in abilities to parse data formats and extract meaningful information from them (here, for example, the pixel resolution and coordinate reference system).

```

1  ~~~joe
2  -input jrc_raster : Geo.Raster
3  +input jrc_raster : Geo.Raster(
4    +   crs="WGS 84",
5    +   pixel_size=(0.000269494585236, -0.000269494585236)
6  +)
7  ...
8  ~~~
9
10 ~~~joe-example
11 -example jrc_raster =
12 -   "/examples/JRC_TMF_AnnualChange_v1_1990_AFR_ID16_S10_E10.tif"
13 +example jrc_raster : mAVUWIDM4vm1PUMXz0IFJhs3waGRTqIi4T0JNeSr0uSAjmPOS =
14 +   "/examples/JRC_TMF_AnnualChange_v1_1990_AFR_ID16_S10_E10.tif"
15 ~~~

```

Listing 11: Joe suggesting to promote back better type information.

Version Control Specifications change. They incorporate new methodologies, fix existing ones and add extra clarification for readers. These changes are often poorly supported in existing tools for writing specifications. I propose to embed a notion of version control directly into the abstract syntax tree of Joe via patches. By building language-specific version control into the tool itself, I hope to mitigate the woes of using general-purpose VCS systems like git.

? Question

Should Joe be a general-purpose specification language or domain-specific? Given the early stage I'm in for the PhD, I believe it is important to tackle big problems and readjust as necessary. To this end, I think Joe should be general-purpose with support for building *domain-specific specification languages* directly in the language itself. I believe this to be similar to how Racket is a *language-oriented programming language*, to quote Matthew Flatt in *Creating Languages in Racket*: "Since programmers often build task-specific tools, one way to make them more productive is to give them better tool-making tools." [34]. An avenue I would like to explore is making Joe a *specification-oriented specification language*.

4.3. A Computational Wiki



Scenario

Having written up her methodology in Joe, Alice opens her exploratory analysis in which she added soil type as a new matching variable. Luckily, she was using the Shark shell and can export her history into a temporary markdown file. She sees some of her own annotations from when she ran commands and the shell has done an admirable job linking commands together in a way that describes how the data flowed from command to command. Her exploratory programming was worth it. Alice patches some of the mistakes and over-approximations and begins running everything locally. The web view helps get the dependency graph just right. Happy with the description of her analysis, she uploads the file to her group's computational wiki along with some of the data already computed locally. She shares this with her colleagues along with the specification.

Shark is a tool for managing a *computational wiki*. A term coined by Prof. Anil Madhavapeddy, a computational wiki is a means for organising information in a reproducible, accessible and *eventually* up to date way. I propose formalising and building a computational wiki to host key scientific research and results. As it pertains to this thesis, the computational wiki will act an environment for the specification language. This is an important clarification as so far Shark has been a tool for executing implementations – I would like to shift my research focus from the execution of implementations to the writing and creation of methodologies.

There is a clear distinction between the computational wiki and the tool used to manage it (*Shark*). This distinction is crucial to the success of the project. I'm reminded of, for example, OCaml package management and the distinction between the curated, thoroughly tested collection of metadata about packages (called the opam-repository) and the instantiation of a *tool* for managing and using that data (the OCaml package manager opam). Arguably, the metadata repository and its organisation *is* OCaml package management.

Concretely, I believe the direction of research for the computational wiki should be to deliver the following five technical details (please refer to the gap analysis Figure 6).

Versioning In the context of important results in climate science there is a need to remain up to date with the latest research **and** to preserve previous attempts and versions for retrospective analysis and reasoning. Doing so for large datasets is not without its challenges.

Controlling Information Flow Having a secure mechanism for controlling how information flows across the wiki is crucial to protecting valuable, private information but still providing those with the correct clearance access to that data.

Tracking Data Lineage Data must describe how it came to be. This is too common of a pitfall in computational science to not address it directly.

Discovering Data With the vision for the computational wiki to be large and varied in format (code, datasets, raster images etc.), making data easily indexed and discoverable is needed to make it an attractive platform to scientists.

Garbage Collection Resources are not infinite and sometimes data is only meant to be temporary, which data to keep and to lose likely needs a social metric to understand its relevance to the scientific community at large.

It would seem to also make sense to make the computational wiki *self-describing* – it should be a standalone artefact that needs no additional tool or program in order to reconstruct itself, albeit slowly, from scratch should that be necessary.

4.4. Timeline

The thesis is thus separated into three sections: groundwork to inform uncertainty within climate science, a core specification language and an environment for that language: the computational wiki. These three components have been roughly scheduled across the next two years in Figure 9.

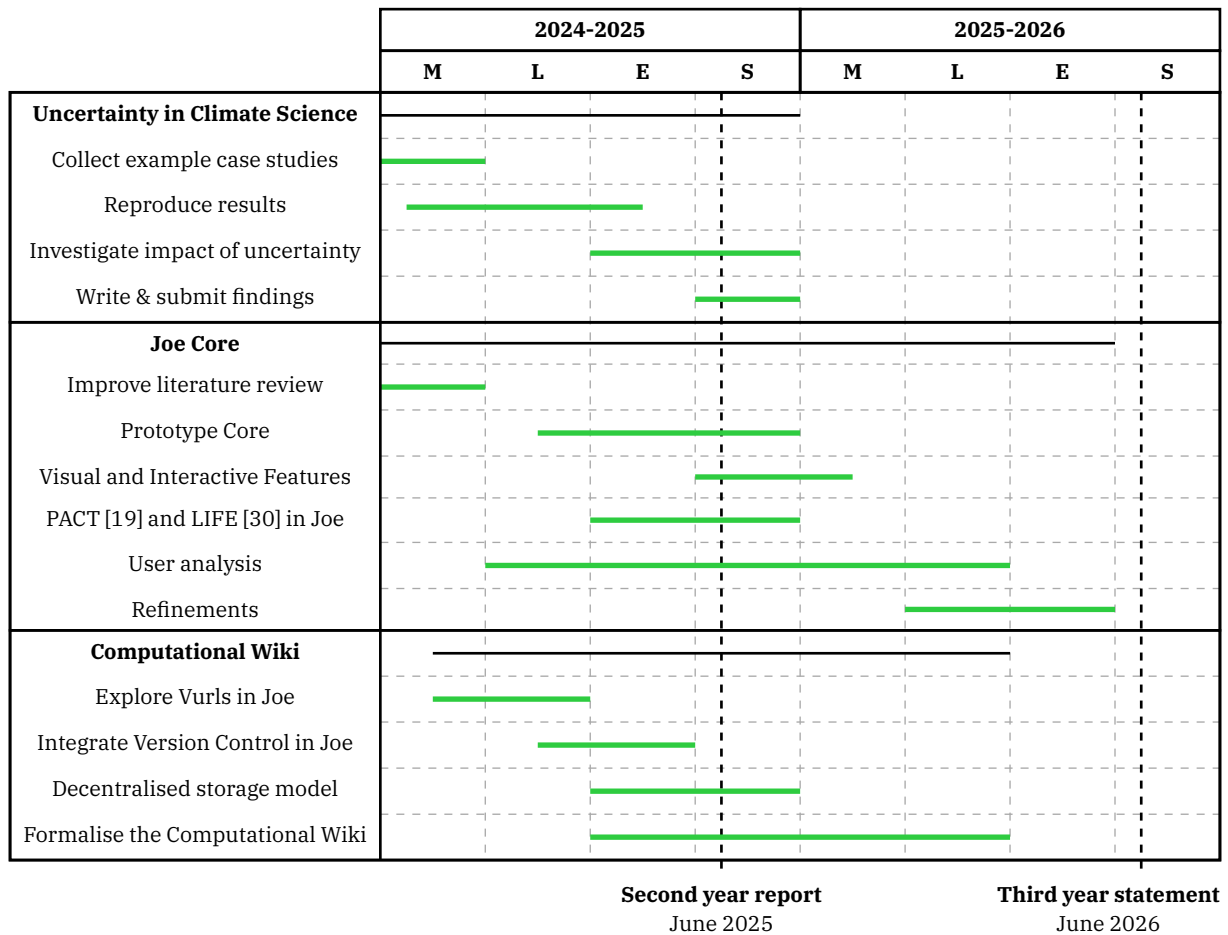


Figure 9: A proposed timeline for research over the next two years.

In terms of developing the specification language Joe, we are in a fortunate position to work alongside ecologists, zoologists and geographers in the Cambridge Conservation Initiative. One possibility is to develop Joe under a similar methodology as the one described in *PLIERS* (Programming Language Iterative Evaluation and Refinement System) which integrates qualitative and quantitative methods to design programming languages [26]. More thought is needed about the exact methodology and the research ethics that apply.

A Uncertainty at Scale

Uncertainty at scale: how CS hinders climate research

Patrick Ferris, Michael Dales, Tom Swinfield, Sadiq Jaffer, Srinivasan Keshav, Anil Madhavapeddy
Departments of Computer Science & Technology and Zoology, University of Cambridge, UK

Computer science is a powerful tool for enabling data-driven advances in global ecology and conservation. However the amplification cuts two ways, as mechanisation can also compound problems inherent with just how uncertain [13] anything to do with natural ecosystems are! Species habitat datasets are uncertain, local observations are uncertain, the resulting inferences about species distributions are uncertain, side-effects from interventions are uncertain; conservation action has evolved to take this into account [17]. Computer science when applied without consideration of these factors can amplify the uncertainty by running ever-larger datasets through increasingly complex data pipelines and algorithms, all built upon wobbly foundations. What exact version of the dataset is being used? What exact version of the dataset *did* you use? What assumptions went into generating that dataset? What libraries, system dependencies and environment variables were used to calculate the results?

In this talk, we first segment sources of uncertainty across ecological data sources (§1) and computation over them (§2), and then reflect how these uncertainties impact ecological research and how we might cleanly bound the uncertainty for future conservation research (§3).

1 UNCERTAINTY IN DATA

As early as 2013, ecologists have been calling on their colleagues “...to treat data as an enduring product of research, not just a precursor to publication” [12]. This narrative has played out in computer science with version control systems (VCS) for software management, or for machine learning with *Weights and Biases* for AI experiment tracking and *HuggingFace* for AI model versioning [2, 3]. However, pivotal datasets within ecology are versioned ambiguously, leading to differing end results from seemingly identical inputs.

One example is the Tropical Moist Forest (TMF) dataset by the European Commission’s Joint Research Centre [18], which calculates forest cover worldwide from satellite observations. TMF historic data is upgraded as new algorithms and analysis become available (which is good practise), but the means by which the data is published does not make these updates to historically available data obvious. Once updated, earlier versions on which other calculations were made are no longer easily available, hugely impacting research reproducibility [7]. Table 1 shows some differences partially caused by “Improvements and corrections of errors in the Annual Change collection in the sequence of values for deforestation of old regrowth forest...” [4]. Whilst they seem

Land Use Class Proportions (%) in 2020

	JRC 2021	JRC 2022	Difference
Undisturbed	74.83	74.71	-0.12
Degraded	5.07	5.16	0.09
Deforested	7.49	7.39	-0.10
Regrowth	0.83	1.74	0.91
Water	2.11	1.93	-0.18
Other	9.67	9.07	-0.60

Table 1: The proportions of difference Land Use Classes (LUC) in the Amazon basin as calculated from two JRC datasets for the same year.

small, these proportions represent some 6.7 million squared kilometres area of the Amazon rainforest basin¹! Without access to the original 2021 JRC release, these differences propagate silently through further downstream research.

2 UNCERTAINTY IN CODE

Once ingested into a data pipeline, the data’s provenance is usually lost, as untracked inputs are combined into derived datasets whose origins only the operator may remember, as operating systems do not automatically track such things. Ecological data are often transformed using programming languages like Python and R, but the ecological community has yet to adopt a robust culture of openly publishing their code alongside their methodologies. One reason for this, that Mislán et al. recognise, is that ecologists “...may not be aware of the steps needed to archive code...” [16]. The tools for versioning and archiving code are not easily accessible for ecologists, and whilst the number of ecological journals requiring or encouraging code to be shared has increased, most authors still do not adhere to these requirements [8].

Even when care is taken, without reproducible artefacts (e.g., using a system like Nix [9]), the sheer multiplicity of factors that can change the final results is unmanageable within conventional operating systems. For example, consider the popular geospatial data manipulation library GDAL [1]. The image in figure 1 is the difference between data derived with *the same* command-line but with GDAL 3.2 and GDAL 3.3.

The low availability of source code only exacerbates the reproducibility story when combined with the data versioning issues (§1). Whilst some computer science research has offered accessible versioning of large datasets, like DataHub, it is not habitually used within ecological research [6].

¹Source code: <https://github.com/carboncredits/jrc-diff>



Figure 1: A map showing the difference in terrain ruggedness index (TRI) as calculated by the `gdaldem tri` command between GDAL versions 3.2 and 3.3. The difference would ideally be zero (all black) but includes pixels as different as 372 metres.

One platform has cornered the remote sensing, ecological research market and that is Google Earth Engine (GEE) [11]. GEE has enabled many ecologists to produce useful analysis and datasets that may have been difficult to achieve otherwise. For example, the global map of travel time to healthcare facilities which can be used as a proxy for “remoteness” in ecological analysis [19]. However, the longevity and endurance of the platform is tied to the whims of a large corporation that has a record of closing down non-core products [10]. Also, many datasets in GEE have been processed to make them easier to consume, but the methodology by which they’ve been processed is not published, preventing easy migration away from GEE once datasets are adapted.

3 UNCERTAINTY IN ECOLOGY

The lack of software engineering skills and tools impacts the rigour and quality of ecological research. The analysis can explode in both space and time: even a relatively simple species area of habitat analysis requires hundreds of gigabytes of raster data to be analysed per species, with typically many thousands of species being processed [14], potentially requiring days of compute on high-end hardware.

Unoptimised code limits the analysis that can be performed due to the time or resources it would require. This is important for sensitivity analysis for methodologies with a large number of inputs and possible configurations. In the context of Merton’s notion of “specified ignorance” (the knowledge that is not yet known but ought to be) these issues due to computer science should be flagged as sources of ignorance and therefore fruitful research directions for

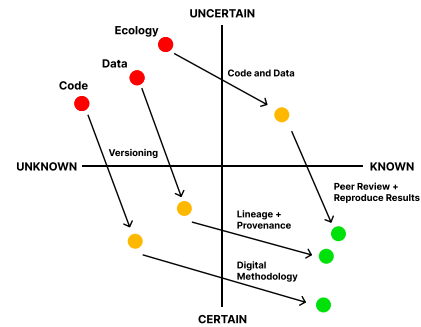


Figure 2: Shifting unknown code and data certainties in order to allow computation ecological research to be meaningfully peer-reviewed and reproduced.

others [15]. This idea is explored in terms of known and unknown certainties in Figure 2. Code and data are bounded via versioning, provenance and lineage tracking and shifting to digital methodologies. This is not dissimilar to the use of *continuous integration* systems for managing production-level code. Only once code and data are bounded can we hope to return to applying the peer review process to reproduce results and have known, certain ecological research.

The computer science aspects must be dealt with first. In recent years, we have seen the rise of the *Research Software Engineer* [20]; the primary role of the research software engineer is to provide software skills to another discipline. Computer science and software can be thought of as a new instrument for experimental scientists. In their paper on the research software engineer, Baxter et al. argue a key difference between more traditional scientific methods and the newer computational ones are that “...no-one with the same casual attitude to experimental instrumentation as many researchers have to code would be allowed anywhere near a lab” [5]. They later cite one reason for the short supply of research-focused software engineers is the lack of “institutional homes and career progression paths for their work” which could explain why software is treated differently to more traditional instruments used in experiments.

The scientific method in the computational age has become vastly more complicated. Computer science has thus far failed to deliver the tools to scientists to easily bound this complexity and reduce uncertainty in their analysis. Uncertainty within methodologies is difficult to correct. However, ensuring that the computing foundations upon which ecologists build their work minimises and tracks uncertainty is achievable, and this must be a priority for our field.

REFERENCES

- [1] GDAL. <https://gdal.org/>.
- [2] Experiment tracking with weights and biases. <https://www.wandb.com/>, 2020. Software available from wandb.com.
- [3] Huggingface: We are on a mission to democratize good machine learning, one commit at a time. <https://huggingface.co>, 2023.
- [4] List of updates integrated in the TMF 2022 products. <https://forobs.jrc.ec.europa.eu/TMF/data#update>, 2023.
- [5] BAXTER, R., CHUE HONG, N., GORISSEN, D., HETHERINGTON, J., AND TODOROV, I. The research software engineer. Digital Research 2012 ; Conference date: 10-09-2012 Through 12-09-2012.
- [6] BHARDWAJ, A., BHATTACHERJEE, S., CHAVAN, A., DESHPANDE, A., ELMORE, A. J., MADDEN, S., AND PARAMESWARAN, A. G. Datahub: Collaborative data science & dataset version management at scale. *arXiv preprint arXiv:1409.0798* (2014).
- [7] CLARK, B., DESHANE, T., DOW, E., EVANCHIK, S., FINLAYSON, M., HERNE, J., AND MATTHEWS, J. Xen and the art of repeated research. In *Proceedings of the FREENIX* (2004), pp. 135–144.
- [8] CULINA, A., VAN DEN BERG, I., EVANS, S., AND SÁNCHEZ-TÓJAR, A. Low availability of code in ecology: A call for urgent action. *PLOS Biology* 18, 7 (07 2020), 1–9.
- [9] DOLSTRA, E., AND DE JONGE, M. Nix: A safe and Policy-Free system for software deployment. In *18th Large Installation System Administration Conference (LISA 04)* (Atlanta, GA, Nov. 2004), USENIX Association.
- [10] GERKEN, T. Gamers say goodbye to google’s stadia as platform shuts, 2023. Accessed on October 23, 2023.
- [11] GORELICK, N., HANCHER, M., DIXON, M., ILYUSHCHENKO, S., THAU, D., AND MOORE, R. Google earth engine: Planetary-scale geospatial analysis for everyone. *re-mote sensing of environment*, 202, 18–27, 2017.
- [12] HAMPTON, S. E., STRASSER, C. A., TEWKSBURY, J. J., GRAM, W. K., BUDDEN, A. E., BATCHELLER, A. L., DUKE, C. S., AND PORTER, J. H. Big data and the future of ecology. *Frontiers in Ecology and the Environment* 11, 3 (2013), 156–162.
- [13] LEWANDOWSKY, S., BALLARD, T., AND PANCOST, R. D. Uncertainty as knowledge. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 373, 2055 (2015), 20140462.
- [14] LUMBIERRES, M., DAHAL, P., SORIA, C., DI MARCO, M., BUTCHART, S., DONALD, P., AND RONDININI, C. Area of habitat maps for the world’s terrestrial birds and mammals. *Sci Data* (2022).
- [15] MERTON, R. K. Three fragments from a sociologist’s notebooks: Establishing the phenomenon, specified ignorance, and strategic research materials. *Annual Review of Sociology* 13, 1 (1987), 1–29.
- [16] MISLAN, K., HEER, J. M., AND WHITE, E. P. Elevating the status of code in ecology. *Trends in Ecology & Evolution* 31, 1 (2016), 4–7.
- [17] SUTHERLAND, W. J., Ed. *Transforming Conservation: A Practical Guide to Evidence and Decision Making*. Open Book Publishers, Dec. 2022.
- [18] VANCUTSEM, C., ACHARD, F., PEKEL, J.-F., VIEILLEDENT, G., CARBONI, S., SIMONETTI, D., GALLEGO, J., ARAGÃO, L. E. O. C., AND NASI, R. Long-term (1990–2019) monitoring of forest cover changes in the humid tropics. *Science Advances* 7, 10 (2021), eabe1603.
- [19] WEISS, D. J., NELSON, A., VARGAS-RUIZ, C. A., GLIGORIĆ, K., BAVADEKAR, S., GABRILOVICH, E., BERTOZZI-VILLA, A., ROZIER, J., GIBSON, H. S., SHEKEL, T., KAMATH, C., LIEBER, A., SCHULMAN, K., SHAO, Y., QARKAKHIJA, V., NANDI, A. K., KEDDIE, S. H., RUMISHA, S., AMRATIA, P., ARAMBEPOLA, R., CHESTNUTT, E. G., MILLAR, J. J., SYMONS, T. L., CAMERON, E., BATTLE, K. E., BHATT, S., AND GETHING, P. W. Global maps of travel time to healthcare facilities. *Nature Medicine* 26, 12 (Dec. 2020), 1835–1838.
- [20] WOOLSTON, C. Why science needs more research software engineers, 2022.

B Uncertainty in Code using Shark

The following code is from a repository of Shark examples.

Uncertainty in Code <uncertainty-in-code>

Uncertainty in computational, ecological methodologies can arise from the environment. This includes the installed libraries and packages on the machine running any analysis.

For example, consider calculating the *_Terrain Ruggedness Index_* (TRI) using GDAL. First we grab two versions of GDAL. The first is ****v3.2****.

```
~~~shark-build:v3.2
((from "osgeo/gdal:ubuntu-small-3.2.0")
 (run (shell "mkdir /data")))
~~~
```

And the second is ****v3.3****.

```
~~~shark-build:v3.3
((from "osgeo/gdal:ubuntu-small-3.3.0")
 (run (shell "mkdir /data")))
~~~
```

The dataset

We will use the SRTM dataset as an example of what can go wrong.

```
~~~shark-run:v3.2:bdcc35b366eb94eb09247a926a6f7d742645115161c16a7b2d0f87651d3ef209
curl -s https://srtm.csi.cgiar.org/wp-content/uploads/files/srtm_5x5/TIFF/srtm_34_11.
zip > /data/srtm_34_11.zip
~~~
```

We can then unzip and copy the tile out of the downloaded elevation map.

```
~~~shark-run:v3.2:ccf6bf2161b7495236ba8aa8c78f9fcdb844b551d06d33f525573d30ae30e73c
unzip /shark/bdcc35b366eb94eb09247a926a6f7d742645115161c16a7b2d0f87651d3ef209/
srtm_34_11.zip -d /data/srtm_34_11-raw
Archive: /shark/bdcc35b366eb94eb09247a926a6f7d742645115161c16a7b2d0f87651d3ef209/
srtm_34_11.zip
  inflating: /data/srtm_34_11-raw/readme.txt
  inflating: /data/srtm_34_11-raw/srtm_34_11.hdr
  inflating: /data/srtm_34_11-raw/srtm_34_11.tfw
  inflating: /data/srtm_34_11-raw/srtm_34_11.tif
```

```
sha256sum /data/srtm_34_11-raw/srtm_34_11.tif
c512ed327e814864a56af290d86a93243d26d6098c6968b6f43059cc7c68dc5e /data/srtm_34_11-
raw/srtm_34_11.tif
```

~~~

### ## Preprocessing

Before doing the analysis, we need to warp the raw image into a suitable CRS.

```
~~~shark-run:v3.3:67bd64211719d28e4d801527fb7be7f0ee39d2ae6e5caf49ea2cff54314415d5
gdalwarp -s_srs "EPSG:4326" -t_srs '+proj=utm +zone=29 +datum=WGS84 +units=m +no_defs'
-oF GTIFF /shark/ccf6bf2161b7495236ba8aa8c78f9fcdb844b551d06d33f525573d30ae30e73c/
srtm_34_11-raw/srtm_34_11.tif /data/srtm_34_11.tif
```

```

Creating output file that is 6037P x 6058L.
Processing /shark/ccf6bf2161b7495236ba8aa8c78f9fcdb844b551d06d33f525573d30ae30e73c/
srtm_34_11-raw/srtm_34_11.tif [1/1] : 0Using internal nodata values (e.g. -32768)
for image /shark/ccf6bf2161b7495236ba8aa8c78f9fcdb844b551d06d33f525573d30ae30e73c/
srtm_34_11-raw/srtm_34_11.tif.
Copying nodata values from source /
shark/ccf6bf2161b7495236ba8aa8c78f9fcdb844b551d06d33f525573d30ae30e73c/srtm_34_11-raw/
srtm_34_11.tif to destination /data/srtm_34_11.tif.
...10...20...30...40...50...60...70...80...90...100 - done.

```

~~~

## ## Calculate TRI

There is a command line tool that is packaged with GDAL to calculate TRI.

We can do this with tool packaged with GDAL **vv3.2vv**.

```

~~~shark-run:v3.2:1397a726f68551fc75e4b14b1454782f698e9c7b3c4bb889558040372de43fd3
gdaldem
tri -q /shark/67bd64211719d28e4d801527fb7be7f0ee39d2ae6e5caf49ea2cff54314415d5/
srtm_34_11.tif /data/srtm_34_11-v3.2.tif
~~~

```

And then again for the same tool package with GDAL **vv3.3vv**.

```

~~~shark-run:v3.3:4c12ad4f4d6d47ef5e8922b63887161e18604d62464e081ce6290804e8934d2e
gdaldem
tri -q /shark/67bd64211719d28e4d801527fb7be7f0ee39d2ae6e5caf49ea2cff54314415d5/
srtm_34_11.tif /data/srtm_34_11-v3.3.tif
~~~

```

First we perform a test where the difference between two rasters should be nothing.

```

~~~shark-run:v3.2:fad67d1e91fe610853d699fc46fa2a346c0dcf8ab62b1d4401ea2feb7ab71368
gdal_calc.py --quiet --cal="A - B"
-A /shark/1397a726f68551fc75e4b14b1454782f698e9c7b3c4bb889558040372de43fd3/srtm_34_11-
v3.2.tif -B /shark/1397a726f68551fc75e4b14b1454782f698e9c7b3c4bb889558040372de43fd3/
srtm_34_11-v3.2.tif --outfile=/data/srtm-tri-no-diff.tif
gdalinfo -stats /data/srtm-tri-no-diff.tif | sed -n '/Band 1 Block/, $p'
Band 1 Block=6037x1 Type=Float32, ColorInterp=Gray
  Minimum=0.000, Maximum=0.000, Mean=0.000, StdDev=0.000
  NoData Value=3.40282346600000016e+38
  Metadata:
    STATISTICS_MAXIMUM=0
    STATISTICS_MEAN=0
    STATISTICS_MINIMUM=0
    STATISTICS_STDDEV=0
    STATISTICS_VALID_PERCENT=39.57

```

~~~

No we do the same but for the two TRIs generate with GDAL **vv3.2vv** and **vv3.3vv**.

```

~~~shark-run:v3.2:3dbbd56760c5168b0b1d0594b09b299cc99934453eb1b05bf207233c547f2825
gdal_calc.py --quiet --cal="A - B"
-A /shark/1397a726f68551fc75e4b14b1454782f698e9c7b3c4bb889558040372de43fd3/srtm_34_11-
v3.2.tif -B /shark/4c12ad4f4d6d47ef5e8922b63887161e18604d62464e081ce6290804e8934d2e/
srtm_34_11-v3.3.tif --outfile=/data/srtm-tri-diff.tif
gdalinfo -stats /data/srtm-tri-diff.tif | sed -n '/Band 1 Block/, $p'

```

```
Band 1 Block=6037x1 Type=Float32, ColorInterp=Gray
  Minimum=-672.854, Maximum=0.000, Mean=-15.297, StdDev=14.790
  NoData Value=3.40282346600000016e+38
  Metadata:
    STATISTICS_MAXIMUM=0
    STATISTICS_MEAN=-15.297075732539
    STATISTICS_MINIMUM=-672.85357666016
    STATISTICS_STDDEV=14.790352827817
    STATISTICS_VALID_PERCENT=39.57
```

~~~

The diff image can then be published locally by [Shark](<https://github.com/quantifyearth/shark>).

~~~shark-publish

```
/obuilder-zfs/
result/3dbbd56760c5168b0b1d0594b09b299cc99934453eb1b05bf207233c547f2825/.zfs/snapshot/
snap/rootfs/data/srtm-tri-diff.tif
```

~~~

## Bibliography

- [1] About RFCs.
- [2] RFC Errata Report >> RFC Editor.
- [3] Typst: Compose Papers Faster.
- [4] Racket.
- [5] Pollen: The Book Is a Program.
- [6] Geoparquet.
- [7] QGIS.
- [8] RStudio.
- [9] Observable: Build Expressive Charts and Dashboards with Code.
- [10] R: The R Project for Statistical Computing.
- [11] MATLAB.
- [12] Unison | A Friendly, Statically-Typed, Functional Programming Language from the Future *c* · Unison Programming Language.
- [13] Observable Framework.
- [14] 2024. Ahrefs/Atd.
- [15] 2024. Python.
- [16] Wil Van Der Aalst and Wil van der Aalst. 2016. *Data Science in Action*. Springer.
- [17] Paulo Sérgio Almeida. 2023. Approaches to Conflict-Free Replicated Data Types. (October 2023).
- [18] Antonio Badia. 2021. *SQL for Data Science*. Springer.
- [19] Andrew Balmford, David Coomes, Michael Dales, Patrick Ferris, James Hartup, Sadiq Jaffer, Srinivasan Keshav, Miranda Lam, Anil Madhavapeddy, Robin Message, and et al. 2023. PACT Tropical Moist Forest Accreditation Methodology v2.0. *Cambridge Open Engage* (2023). <https://doi.org/10.33774/coe-2023-g584d-v5>
- [20] Mike Bishop. 2022. *HTTP/3*. <https://doi.org/10.17487/RFC9114>
- [21] James Bornholt, Rajeev Joshi, Vytautas Astrauskas, Brendan Cully, Bernhard Kragl, Seth Markle, Kyle Sauri, Drew Schleit, Grant Slatton, Serdar Tasiran, Jacob Van Geffen, and Andrew Warfield. 2021. Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (Sosp '21)*, 2021. Association for Computing Machinery, New York, NY, USA, 836–850. <https://doi.org/10.1145/3477132.3483540>
- [22] Harvey Brooks. 1994. The Relationship between Science and Technology. *Research policy* 23, 5 (1994), 477–486.
- [23] H. Butler, M. Daly, A. Doyle, Sean Gillies, T. Schaub, and Stefan Hagen. 2016. *The GeoJSON Format*. <https://doi.org/10.17487/RFC7946>
- [24] John M. Chambers. 2014. Object-Oriented Programming, Functional Programming and R. *Statistical Science* 29, 2 (2014), 167–180. <https://doi.org/10.1214/13-STS452>
- [25] Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. 2016. Reprozip: Computational Reproducibility with Ease. In *Proceedings of the 2016 International Conference on Management of Data*, 2016. 2085–2088.
- [26] Michael Coblenz, Gauri Kambhatla, Paulette Koronkevich, Jenna L. Wise, Celeste Barnaby, Joshua Sunshine, Jonathan Aldrich, and Brad A. Myers. 2021. PLIERS: A Process That Integrates User-Centered Methods into Programming Language Design. *ACM Trans. Comput.-Hum. Interact.* 28, 4 (July 2021), 1–53. <https://doi.org/10.1145/3452379>
- [27] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Communications of The Acm* 13, 6 (June 1970), 377–387. <https://doi.org/10.1145/362384.362685>
- [28] Colby Hopkins. 2023. The History of Amazon and Its Rise to Success.
- [29] Cynthia Dwork. 2006. Differential Privacy. In *International Colloquium on Automata, Languages, and Programming*, 2006. Springer, 1–12.
- [30] Alison Eyres, Thomas Ball, Michael Dales, Tom Swinfield, Andy Arnell, Daniele Baisero, América Paz Durán, Jonathan Green, Rhys E. Green, Anil Madhavapeddy, and Andrew Balmford. 2024. LIFE: A Metric for Quantitatively Mapping the Impact of Land-Cover Change on Global Extinctions. <https://doi.org/10.33774/coe-2023-gpn4p-v4>

- [31] Patrick Ferris, Michael Dales, Sadiq Jaffer, Amelia Holcomb, Eleanor Toye Scott, Thomas Swinfield, Alison Eyres, Andrew Balmford, David Coomes, Srinivasan Keshav, and Anil Madhavapeddy. 2024. Planetary Computing for Data-Driven Environmental Policy-Making.
- [32] Patrick Ferris, Michael Dales, Tom Swinfield, Sadiq Jaffer, Srinivasan Keshav, and Anil Madhavapeddy. 2024. Uncertainty at Scale: How CS Hinders Climate Science.
- [33] Heather Flanagan. 2019. *Fifty Years of RFCs*. <https://doi.org/10.17487/RFC8700>
- [34] Matthew Flatt. 2012. Creating Languages in Racket. *Communications of the ACM* 55, 1 (2012), 48–56.
- [35] James R Groff, Paul N Weinberg, and Andrew J Oppel. 2002. *SQL: The Complete Reference*. McGraw-Hill/Osborne.
- [36] Joel M. Halpern, Leslie Daigle, Olaf Kolkman, and IAB. 2016. *RFC Streams, Headers, and Boilerplates*. <https://doi.org/10.17487/RFC7841>
- [37] David Harel. 1987. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming* 8, 3 (1987), 231–274. [https://doi.org/10.1016/0167-6423\(87\)90035-9](https://doi.org/10.1016/0167-6423(87)90035-9)
- [38] John Gruber. Markdown.
- [39] Karissa Rae McKelvey, Scott Jenson, Eileen Wagner, Blaine Cook, and Martin Kleppmann. 2023. Upwelling: Combining Real-Time Collaboration with Version Control for Writers.
- [40] Mary Beth Kery, Amber Horvath, and Brad A Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *CHI*, 2017. 3025453–3025626.
- [41] Martin Kleppmann, Adam Wiggins, Peter Van Hardenberg, and Mark McGranaghan. 2019. Local-First Software: You Own Your Data, in Spite of the Cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2019. 154–178.
- [42] Donald Ervin Knuth. 1984. Literate Programming. *The computer journal* 27, 2 (1984), 97–111.
- [43] Jiangshan Lai, Christopher J Lortie, Robert A Muenchen, Jian Yang, and Keping Ma. 2019. Evaluating the Popularity of R in Ecology. *Ecosphere (Washington, D.C)* 10, 1 (2019), e2567.
- [44] Leslie Lamport. 2002. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. (2002).
- [45] Leslie Lamport. 2009. LaTeX : A Document Preparation System : User's Guide and Reference Manual / Leslie Lamport ; Illustrations by Duane Bibby. *LaTeX : A Document Preparation System : User's Guide and Reference Manual*.
- [46] N.G. Leveson, M.P.E. Heimdahl, H. Hildreth, and J.D. Reese. 1994. Requirements Specification for Process-Control Systems. *IEEE Transactions on Software Engineering* 20, 9 (1994), 684–707. <https://doi.org/10.1109/32.317428>
- [47] Linus Torvalds. 1994. Linux Kernel v1.0.
- [48] Geoffrey Litt, Sarah Lim, Martin Kleppmann, and Peter Van Hardenberg. 2022. Peritext: A Crdt for Collaborative Rich Text Editing. *Proceedings of the ACM on Human-Computer Interaction* 6, CSCW2 (2022), 1–36.
- [49] Stefan Malewski, Michael Greenberg, and Éric Tanter. 2021. Gradually Structured Data. *Proc. ACM Program. Lang.* 5, OOPSLA (October 2021). <https://doi.org/10.1145/3485503>
- [50] Martin Kleppmann. Automerge: CRDTs Meet Version Control.
- [51] Philip Matthews, Jonathan Rosenberg, Dan Wing, and Rohan Mahy. 2008. *Session Traversal Utilities for NAT (STUN)*. <https://doi.org/10.17487/RFC5389>
- [52] Peter J. McCann and Satish Chandra. 2000. Packet Types: Abstract Specification of Network Protocol Messages. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (Sigcomm '00)*, 2000. Association for Computing Machinery, New York, NY, USA, 321–333. <https://doi.org/10.1145/347059.347563>
- [53] Claudio Menghi, Christos Tsiganos, Patrizio Pelliccione, Carlo Ghezzi, and Thorsten Berger. 2019. Specification Patterns for Robotic Missions. *IEEE Transactions on Software Engineering* 47, 10 (2019), 2208–2224.
- [54] Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. 2021. Catala: A Programming Language for the Law. *Proc. ACM Program. Lang.* 5, ICFP (August 2021), 1–29. <https://doi.org/10.1145/3473582>
- [55] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and How to Develop Domain-Specific Languages. *Acm Computing Surveys* 37, 4 (December 2005), 316–344. <https://doi.org/10.1145/1118890.1118892>
- [56] Michael Lucas and Allen Jude. 2015. FreeBSD Mastery: ZFS. *FreeBSD Mastery: ZFS*.

- [57] Mike Bostock. TopoJSON Specification.
- [58] Mark S Miller, Ka-Ping Yee, Jonathan Shapiro, and others. 2003. *Capability Myths Demolished*.
- [59] Raphaël Monat, Aymeric Fromherz, and Denis Merigoux. 2024. Formalizing Date Arithmetic and Statically Detecting Ambiguities for the Law. In *Programming Languages and Systems: 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6–11, 2024, Proceedings, Part II*, 2024. Springer-Verlag, Berlin, Heidelberg, 421–450. [https://doi.org/10.1007/978-3-031-57267-8\\_16](https://doi.org/10.1007/978-3-031-57267-8_16)
- [60] Raphaël Monat, Aymeric Fromherz, and Denis Merigoux. 2024. Formalizing Date Arithmetic and Statically Detecting Ambiguities for the Law (Artifact). <https://doi.org/10.5281/zenodo.10460049>
- [61] Dominic P. Mulligan, Scott Owens, Kathryn E. Gray, Tom Ridge, and Peter Sewell. 2014. Lem: Reusable Engineering of Real-World Semantics. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming (Icfp '14)*, 2014. Association for Computing Machinery, New York, NY, USA, 175–188. <https://doi.org/10.1145/2628136.2628143>
- [62] Andrew C. Myers and Barbara Liskov. 1997. A Decentralized Model for Information Flow Control. *SIGOPS Oper. Syst. Rev.* 31, 5 (October 1997), 129–142. <https://doi.org/10.1145/269005.266669>
- [63] OGC. OGC GeoTIFF Standard.
- [64] Cyrus Omar, Andrew Blinn, and Andrew Bandukwala. Toward a Live, Rich, Composable, and Collaborative Planetary Compute Engine.
- [65] Cyrus Omar, David Moon, Andrew Blinn, Ian Voysey, Nick Collins, and Ravi Chugh. 2021. Filling Typed Holes with Live GUIs. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2021)*, June 2021. Association for Computing Machinery, New York, NY, USA, 511–525. <https://doi.org/10.1145/3453483.3454059>
- [66] Cyrus Omar, Ian Voysey, Ravi Chugh, and Matthew A. Hammer. 2019. Live Functional Programming with Typed Holes. *Proc. ACM Program. Lang.* 3, POPL (January 2019). <https://doi.org/10.1145/3290327>
- [67] Open Geospatial Consortium. Well-Known Text Representation of Coordinate Reference Systems.
- [68] Elizaveta Pertseva, Melinda Chang, Ulia Zaman, and Michael Coblenz. 2024. A Theory of Scientific Programming Efficacy. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*, April 2024. Association for Computing Machinery, New York, NY, USA, 1–12. <https://doi.org/10.1145/3597503.3639139>
- [69] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. 2016. Foundations of JSON Schema. In *Proceedings of the 25th International Conference on World Wide Web*, 2016. 263–273.
- [70] Hannah Potter, Ardi Madadi, René Just, and Cyrus Omar. 2022. Contextualized Programming Language Documentation. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2022)*, 2022. Association for Computing Machinery, New York, NY, USA, 1–15. <https://doi.org/10.1145/3563835.3567654>
- [71] Stephen M Powers and Stephanie E Hampton. 2019. Open Science, Reproducibility, and Transparency in Ecology. *Ecological applications* 29, 1 (2019), e1822.
- [72] Karthik Ram. 2013. Git Can Facilitate Greater Reproducibility and Increased Transparency in Science. *Source Code for Biology and Medicine* 8, 1 (February 2013), 7. <https://doi.org/10.1186/1751-0473-8-7>
- [73] Russ Cox. The Principles of Versioning in Go.
- [74] Mary Shaw. 2022. Myths and Mythconceptions: What Does It Mean to Be a Programming Language, Anyway?. *Proceedings of the ACM on Programming Languages* 4, HOPL (2022), 1–44.
- [75] J Michael Spivey. 1988. *Understanding Z: A Specification Language and Its Formal Semantics*. Cambridge University Press.
- [76] \relax KR Srinath. 2017. Python—the Fastest Growing Programming Language. *International Research Journal of Engineering and Technology* 4, 12 (2017), 354–357.
- [77] Tim Storer. 2017. Bridging the Chasm: A Survey of Software Engineering Practice in Scientific Programming. *Acm Computing Surveys* 50, 4 (August 2017). <https://doi.org/10.1145/3084225>
- [78] Jeff Tollefson. 2019. Humans Are Driving One Million Species to Extinction. *Nature* 569, 7755 (May 2019), 171. <https://doi.org/10.1038/d41586-019-01448-4>
- [79] Jeff Tollefson. 2022. Carbon Emissions Hit New High: Warning from COP27. *Nature* (November 2022). <https://doi.org/10.1038/d41586-022-03657-w>
- [80] Unison Team. Unison.

- [81] C. Vancutsem, F. Achard, J.-F. Pekel, G. Vieilledent, S. Carboni, D. Simonetti, J. Gallego, L. E. O. C. Aragão, and R. Nasi. 2021. Long-Term (1990–2019) Monitoring of Forest Cover Changes in the Humid Tropics. *Science Advances* 7, 10 (2021), eabe1603. <https://doi.org/10.1126/sciadv.abe1603>
- [82] W3C Authors. World Wide Web Consortium History.
- [83] Sonja Mei Wang, Kristen M Scott, Margarita Artemenko, Milagros Miceli, and Bettina Berendt. 2023. "We Try to Empower Them" - Exploring Future Technologies to Support Migrant Jobseekers. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency (FAccT '23)*, 2023. Association for Computing Machinery, New York, NY, USA, 972–983. <https://doi.org/10.1145/3593013.3594056>
- [84] Hadley Wickham, Mine Çetinkaya-Rundel, and Garrett Grolemund. 2023. *R for Data Science*. "O'Reilly Media, Inc."
- [85] Jane Yen, Ramesh Govindan, and Barath Raghavan. 2021. Tools for Disambiguating Rfcs. In *Proceedings of the Applied Networking Research Workshop (Anrw '21)*, 2021. Association for Computing Machinery, New York, NY, USA, 85–91. <https://doi.org/10.1145/3472305.3472314>
- [86] Jane Yen, Tamás Lévai, Qinyuan Ye, Xiang Ren, Ramesh Govindan, and Barath Raghavan. 2021. Semi-Automated Protocol Disambiguation and Code Generation. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (Sigcomm '21)*, 2021. Association for Computing Machinery, New York, NY, USA, 272–286. <https://doi.org/10.1145/3452296.3472910>
- [87] Eric Zhao, Raef Maroof, Anand Dukkipati, Andrew Blinn, Zhiyi Pan, and Cyrus Omar. 2024. Total Type Error Localization and Recovery with Holes. *Proc. ACM Program. Lang.* 8, POPL (January 2024). <https://doi.org/10.1145/3632910>

## Index of Figures

Figure 1	Common pitfalls in the lifecycle of a computational scientific method .....	3
Figure 2	An excerpt from PACT Tropical Moist Forest Accreditation Methodology [19] .....	5
Figure 3	TMF data from 2008 for a region of Indonesia. The image on the left shows the data produced in 2021. The image on the right shows data in 2022. ....	6
Figure 4	A part of the incremental pipeline for calculating PACT [19], showing how data flows from component to component. Green boxes indicate successfully completed computations, yellow boxes are in the queue to be run and grey are yet to be queued. .	8
Figure 5	An example of a Hazel program evaluation even when it is incomplete .....	13
Figure 6	Gap analysis table .....	16
Figure 7	.....	17
Figure 8	.....	17
Figure 9	A proposed timeline for research over the next two years. ....	22

## Index of Listings

Listing 1	A simple invocation of the <code>gdaldem tri</code> command. ....	7
Listing 2	Simplified <code>Input</code> and <code>Output</code> module for an <code>OBuilder</code> plugin to <code>OCurrent</code> . ....	7
Listing 3	An OCaml program using <code>Vurl</code> to resolve a DOI to a file on the local filesystem .....	9
Listing 4	An example program using the extracted OCaml code from <code>F★</code> to implement a <code>DIFC monad</code> .....	10
Listing 5	Format of STUN Message Header .....	12
Listing 6	A snippet of Pollen markup .....	13
Listing 7	An example of Catala describing a section of the US Tax Code [54] .....	14
Listing 8	An example of a geospatial specification written with Joe .....	18
Listing 9	Joe suggesting to promote back better type information .....	19
Listing 10	An example of a geospatial specification written with Joe .....	19
Listing 11	Joe suggesting to promote back better type information. ....	19