Whither Problem-Solving Environments?

Matthew Dinmore
Johns Hopkins University
Applied Physics Laboratory
Laurel, MD USA
matthew.dinmore@jhuapl.edu

Abstract

During the 1990s and first decade of the 2000s, problem-solving environments (PSEs) were a topic of research among a community with the vision to create software systems "with all of the computational facilities necessary to solve a target class of problems." Use of the term has since declined, with fewer papers focused on core PSE research topics. What happened? Did we achieve the design vision for PSEs through other means – namely computational notebooks – or is there more to do? In this essay, we explore the history and objectives of PSE research, the rise of computational notebooks, whether they achieve these objectives, and why the time is right to renew our PSE research efforts.

CCS Concepts

•Software and its engineering→Software notations and tools→Development frameworks and environments→Integrated and visual development environments •Computing methodologies→Modeling and simulation→ Simulation support systems→ Simulation environments •Applied computing→Physical sciences and engineering

Keywords

Problem-solving environment, computational notebook, science and engineering, artificial intelligence, modeling, simulation

ACM Reference Format:

Matthew Dinmore. 2023. Whither Problem-Solving Environments? *In Proceedings of the 2023 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '23), October 25--27, 2023, Cascais, Portugal.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3622758.3622883

1 Introduction

The rapid growth of desktop computing in the 1980s offered scientists and engineers access to increasingly powerful hardware and a growing body of specialized software for their work. This, however, came with a price: scientists in domains outside of computer science had to effectively become programmers in order to develop new code or even weave together existing software libraries in support of computational experimentation[1]. The addition of network computing in the Internet era of the 1990s made sharing software and data easier, but further complicated managing it for large-scale experiments across grids of computers. These challenges inspired the development of problem-solving environments (PSEs), integrated suites of software designed to interact with scientists in a language natural to their research domain. By abstracting the process of integrating the underlying software components to realize a computational experimental design, PSEs promise to save time, increase software reusability, and ultimately allow scientists to focus on science.

PSEs enjoyed active attention from a motivated research community for almost twenty years, but after a brief peak, research production focused on PSEs receded. This essay explores some reasons for why this may have happened. Our philosophical approach to this study is one of creating engineering knowledge [42]. To achieve this, we enumerate the key design features for PSEs that emerged from implementation practice, and based on this, ask whether we've achieved those requirements in the form of computational notebooks. If not, what have we learned, where do we stand, and where do we go from here? Whither PSEs?

1.1 What is a PSE?

A PSE can be defined succinctly as "a software system that provides all of the computational facilities necessary to solve a target class of problems" [43]. However, this is arguably a broad definition – a laptop with a software development tool would fit this definition if we don't

[1] As a historical aside, it is generally considered that the availability of desktop computers enabled more people to be involved in "end-user computing" (e.g. [32]). However, one may ask whether this was also true in scientific computing; weren't scientists among the first programmers? Anecdotally, this has been commented on by, for example, Stephen Wolfram, in a recent discussion of the 35th anniversary of Mathematica, where he notes it being one of the first tools to allow scientific end users to express their own problems [52]. This also highlights a parallel history of "human computers" – largely women – who did the actual programming of computers on behalf of scientists. The marked decline of women in the field of computer science after 1985 has been anecdotally linked to the rise of personal computing [18].

Table 1. Key design features of PSEs.

Feature	Houstis, Gallopoulos, Bramley, & Rice (2000) [22]	Shaffer, Watson, Kafura, & Ramakrishnan (2000) [46]	Houstis, Rice (2000) [21]	Cunha (2001) [10]
Problem-domain interaction	**Multi-level abstractions** *"Recognize the pervasive multilevel structure in science and engineering objects, and allow the addition of more detail and precision at all levels"*		**Natural languages** *"The dream here is to develop a language that allows the user to specify an 'outline' of the problem and the associated computations"*	**Higher Degrees of User Interaction** *"Increased flexibility in user and component interaction demands user interfaces at distinct abstraction levels"*
Multidiscipline and collaboration support		**Multidisciplinary support / Collaboration support** *"…support the ability of researchers to combine together to form larger, multidisciplinary teams. In practice, this means that the models from the various disciplines involved should be combinable in some way."*	**Collaboratory problem solving** *"The design of the engine requires that these different domain-specific analyses interact in order to find the final solution"*	**Multidisciplinary Nature of the Applications** *"…the need to support interactions between distinct sub-models, based on multiple heterogeneous and hybrid components"*
Intelligent support throughout the problem-solving process		**Recommender systems** *"A recommender system for a PSE serves as an intelligent front-end and guides the user from a high level description of the problem…"*	**Intelligence in computational science** *"… the task of selecting the best software and the associated algorithmic/hardware parameters for a particular problem or computation is often difficult"*	**Intelligence in PSEs** *"Advisoring, explaining, and expert tools are important to assist the user during the development and execution steps"*
Problem-solving knowledge capture and sharing	**Create knowledge bases for solvers and problems** *"Provide automation of (or help for) construction of the PSE, dynamic selection of a solver, selection of hardware, suitability of output, management of long-term computations"*	**Usage documentation, Preservation of expert knowledge** *"…implicit and explicit documentation for use of the code, specifically with respect to parameters and other inputs. The interface could provide advice on reasonable interactions of parameters, or which submodels to use in particular circumstances. At the PSE creation level, PSE-building tools could provide a convenient mechanism for adding and accessing such documentation."*	**Models of Problem Solving** *"People normally have a large context in mind when they start on problem solving; they use 'fuzzy' sketches and back-of-the-envelope analyses to get started. How can [these] be incorporated into tile PSE? Are there easier and more powerful ways to communicate about scientific problem solving? Problem solving is often an iterative process involving changing specifications, strategies, and goals. How can computer power in information processing be used to track the problem solving process, to organize it for review by the human, and for analysis by the PSE?"*	
Software sharing and reuse	**Reuse legacy software** *"Encapsulation is possible for reasonably designed software provided some key parts of the legacy environment persist"*	**Internet accessibility to legacy codes** *"The initial reason why a computational scientist or engineer approaches our research group is that they would like to make their legacy modeling code Web accessible"*	**Software reuse** *"One of the design objectives of future PSEs will be the use of scalable libraries as building blocks for creating seamless scientific applications … To realize this objective we must develop tools that enhance reuse and enable layered approaches to application development."*	**Software Architectures** *"…the focus is put on the reuse of components and their dynamic modification, relying upon objectoriented and component-based technologies"*
Components, component types and integration	**Plug-and-play** *"…a systematic framework with formally defined interfaces, supporting the dynamic assembly of software components, and keeping the framework open and tailored to the needs of scientific PSEs"*	**Integration Visualization, Optimization** *"While each feature described in this list is important in its own right, the important aspect of a PSE for computational science research such as we have described would be the synergy that should result from integrating these features into a single system"*	**PSE Software Architecture** *"…there is a (large) collection of problem solving components (i.e. a workbench), including those that are needed to solve it. Then, the user must first combine some of these components to form a custom PSE and then apply it to solve the problem at hand."*	**Dynamic Configuration and Coordination Issues** *"…support the modification of components and their interaction patterns… design of abstract patterns of interactions, on the dynamic reconfiguration of software architectures, and on the coordination of distributed systems"*
Experiment management	**Create test beds for components and combinations**	**Experiment Management** *"…have the results of the simulation runs be stored automatically in some systematic way that permits recovery of previous runs along with the parameters that initiated the run"*	**Validation of Computations** *"The validation of computational science PSE results is critical, yet minimal effort is spent by users or PSEs to check that the answers are correct. … Yet the costs of experiments is increasing while the cost of computation is decreasing."*	
Computing architecture and infrastructure		**High-performance computing** *"Often, simulations … require access to significant computing resources, such as a parallel supercomputer or an "information grid" of computing resources. In such cases, the PSE should integrate a computing resource management subsystem…"*	**Netcentric computing** *"…put the PSE technology at the finger tips of any scientist or engineer, anytime and anywhere"*	**Infrastructures for PSEs** *"A PSE should be able to work on top of low-level and middleware layers which provide the services of a meta-level distributed operating system for cluster computing and global computing platforms"*

consider the user or efficiency, for example. The PSE literature provides additional insight into the attributes of PSEs that can help in better understanding what makes them different. At the next level of detail, Houstis and Rice [21] offer the formulation of:

PSE = Natural Language + Solvers + Intelligence + Software Bus

These four components suggest, at a high level, insight into the key features that differentiate a PSE from other computing environments. Here, *natural language* refers to the interaction paradigm and specifically the preference for working in the users' domain-oriented language, while also foreshadowing the possibility of natural language interfaces. *Solvers* is a general reference to the domain-specific software (computational algorithms, models, visualizations) and its findability and accessibility that provides sufficient abstraction to support domain-level problem solving, as well as the ability to share and reuse this software within a given community. *Intelligence* suggests system support to the problem solver in all aspects of their work and interactions with the system, from setting up the problem, formulating the solution in software, provisioning computational resources, and sensemaking around and sharing of results. Finally, the *software bus* evokes the need for a plug-and-play architecture of communicating components that is easy to reconfigure and update as capabilities against a class of problems mature. It is worth comparing this to an earlier rendition of the formula [44], which read:

PSE = User Interface + Libraries + Knowledge Base + Integration

While structurally similar, this evolution suggests a trajectory in thinking about the design and utility of PSEs through research experience with them. The increased specificity from user interfaces to natural language, and from libraries to solvers suggests a shift from thinking about PSEs in the software domain toward a greater emphasis on their place in the problem-solving domain. Also notable is the change from knowledge bases, which imply a passive resource available to the problem solver, to *intelligence*, suggesting a more active engagement between the system and the user[2]. It might be argued, however, that integration is a broadly more useful goal than being overly specific about a "software bus" architecture, which while in vogue at that time, is one among several architectural options for software integration. Further, consider that this evolution took place concurrently with the growth of the internet, institutionalization of software engineering practice – especially around code reuse and systems architecture, which figure prominently in the discussion around the later definition – and the stirrings of an artificial intelligence (AI) "Spring."

Other works dove more deeply into the attributes of PSEs, both as realized at the time and as envisioned, toward identifying gaps and establishing a research agenda. Houstis, et. al. [23] offer an earlier summary in an introduction to several articles on the topic, relating conclusions from PSE-focused workshops sponsored by the National Science Foundation (NSF) in 1991 and 1995. Shaffer, et. al. [46] focus on software infrastructure,

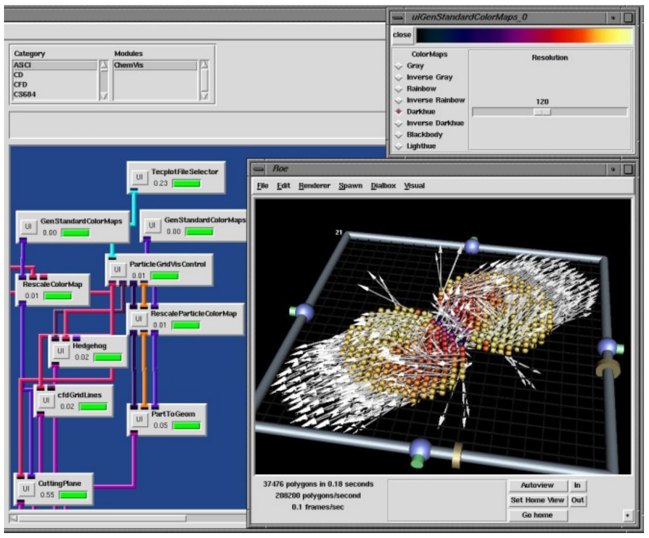

Figure 1. SCIRun PSE showing workflow and results [24].

highlighting challenges PSE users encountered in practice that were not being actively pursued by researchers. Houstis and Rice's 2000 [21] paper, in which the later high-level formulation for PSEs above is found, further decomposes PSE areas of research. Finally, there is Cunha's [10] purposeful review of the requirements for PSEs that identifies key development challenges for next-generation PSEs. Taken together as a sampling of the literature on research into the main features of PSEs, we can draw a more holistic picture of what makes a PSE. Table 1 presents an aggregate list of features in the context of their presentation in each of these papers; only concepts occurring in more than one of these papers are presented, though some interpretive license has been taken in combining differently-worded, but otherwise similar concepts – brief quotations from the original papers are provided to assist the reader in understanding the authors' thinking about these features, though one is referred to these papers for a more thorough discussion. We now briefly describe each area.

1.1.1 Problem Domain Interaction

Closing the gap between domain representations and software is a central objective of a PSE. Enabling interaction with representations that allow the problem solver to work analogically with things they are familiar with is essential [19]. Costabile, et. al. [9] identify two classes of needs for domain experts: (i) parameter setting in a predefined application, and (ii) modifying the software in something akin to programming but "as close as possible to the human," suggesting an aversion to text-based "traditional" programming in favor of end-user programming [32] approaches such as visual programming. Due to the close relationship of PSE research and grid-based computing [14], visual composition environments are common in the PSE literature [15]. General eScience workflow tools with visual composition environments such as Kepler [33] and Taverna [53], as well as PSE-focused tools such as SCIRun

[2] Note that, in this same timeframe, research into mixed-initiative interfaces, e.g. Allen, et. al. 1999 [1] was similarly seeing a peak of interest.

[24], often serve as platforms for PSE research and development. A typical view of a composed workflow and results display from SCIRun is shown in Figure 1, highlighting the two "sides" of a PSE: solution development (workflow composition) and execution [30].

We should note, though, that the domain such representations address is more about that data processing problem – how data collected in an experiment or generated by a simulation – can be processed and analyzed, than the actual scientific problem domain. Other programming approaches to bridging this gap, notably domain-specific languages (DSLs), appear infrequently in the later PSE literature. Earlier efforts to create physical object representations closer to the user's domain of work in general-purpose languages such as LISP [3] and object-oriented environments such as SmallTalk [29] did not ultimately gain traction among PSE developers, with grid computing-focused visual workflow representations dominating in the later years, as evidenced in the prior references. This vision of native problem domain representation in the PSE remains worthy of further work.

1.1.2 Multidiscipline and Collaboration Support

Recognizing that complex problems are often multidisciplinary, requiring experts from several domains, PSEs must enable both natural interaction in the domains of the users and the integration of their contributions to the common solution to the problem. Foster, Papka and Stevens [13] discuss the challenges and trends toward a research agenda in this area, many of which are familiar to our discussion of PSEs, e.g. allowing for flexible interfaces, discovery, sharing and persistence of artifacts, and allocation of computational resources, affirming the importance of collaboration in PSEs. Over a decade later, van der Vet, et. al. [48] reflect that scientists were typically using desktop computers for their work, and while scientific workflow environments such as Taverna assist in "packaging recurrent task sequences in a single environment" to enable *in-silico* experimentation, the broader vision of a truly collaborative environment remained unrealized. Whether the immersive and ambient environments they report on will, as they ask in their discussion, ultimately help, research toward focusing collaboration closer to the problem remains a valid need.

1.1.3 Intelligent Support Throughout the Problem-Solving Process

The variety of intelligent support for the user is imaginably broad, intersecting every phase of the problem-solving process. Early in the process, this would include assistance in stating, scoping and decomposing the problem. Intelligence could then assist in gathering software components, integrating them, or developing novel ones when necessary; this is a significant part of the "recommender system" role envisioned for PSEs. Later, it could assist in deployment to distributed infrastructures, collection and analysis of results, and capture and publication of insights and new knowledge.

1.1.4 Problem-Solving Knowledge Capture and Sharing

There are multiple domains of knowledge relevant to solving a problem in a PSE, including the scientific (or problem domain, in general), software engineering, and computer science [39]. A PSE will ideally support capturing, finding and reusing knowledge across these in an integrated manner.

1.1.5 Software Sharing and Reuse

Similarly, as problem solutions are ultimately realized in software, making that software available and reusable is a key objective. Reuse has been a topic of extensive study in software engineering, though this research has extended in many directions, include several of interest to the features listed here such as problem domain understanding and software components [5]. While the general challenges of reuse apply to PSEs, of particular interest are design for reusability (which can be facilitated by, for example, component frameworks [2]), and maintaining alignment between the problem and the underlying software implementation to make reuse practical [35].

1.1.6 Components, Component Types and Integration

Following from the ideas of representing problem-domain concepts and objects in software and making those representations more shareable and reusable is the notion of encapsulating them as software components. A common component model is required within a system for local integration, but standards become even more important when considering deployment to general grid computing infrastructures [24].

1.1.7 Experiment Management

Problem solving is rarely a linear task; problem solvers iterate on the problem and solution at different points in the process, which makes it desirable to have both a history of what has been tried (and each attempt's results), as well as mechanisms for automating experiments across a range of parameters and collecting the results for analysis [54]. *Computational steering* emerged as an important feature in this, enabling users to monitor and adjust execution of a long-running computations rather than wait until the end to adjust and repeat [30].

1.1.8 Computing Architecture and Infrastructure

As noted previously, the emergence of grid computing and the necessity of enabling scientists to map scientific workflows onto grids became a central aspect of PSEs, with many researchers from the parallel, high-performance and grid computing communities also publishing in PSE-related venues. Arguably, this challenge of data processing became the problem solved by later PSE efforts. The emergence of alternative big data and cloud-based processing paradigms offered other approaches, but in all cases, the need to map a computational problem onto a large-scale computing architecture – which remains largely invisible to the user – and provision the necessary resources continues to be a key requirement of such environments.

1.2 The Summary as a Design Pattern for PSEs

This summarization of features is useful in a number of ways. First, it helps us more thoroughly understand what the PSE research community was pursuing through the emergent properties of PSEs as realized in their body of literature. Second, using this retrospective view of what defines a PSE allows us to evaluate specific instances presented in the literature to determine how representative each is. Third, we can potentially trace the introduction, evolution and degree of objective attainment for each attribute by examining the instances presented, and thereby summarizing a picture of the state of progress against each feature by the PSE research and development community over time. Fourth, we can compare alternatives in the same solution space, which is one of the objectives of this essay. Finally, understanding what makes a PSE allows us to create tools to effectively build PSEs; it is necessary that such tools would include the correct building blocks as well as the appropriate types of "glue" to assemble them in a manner that synergistically results in a capability that achieves the holistic goals of a PSE; in this manner, the summary serves as a design pattern for PSEs.

While several recurring themes emerge in these features – for example the desire to support multidisciplinary, end-user interaction to specify problems, and component-based construction, execution, sharing and reuse of computational solutions – some other areas surprisingly do not. For example, data access and the challenges of "big data" are rarely addressed. This may be because much of this work occurred before the big data era was fully underway. Emphasis on integration of models and simulations that generate data internally at compute time rather than relying on existing data may be another. Or, the close relationship between PSE research and grid computing, where principal concepts are distributed computing and bringing compute to the data may be factors. In any case, with the advent of the data-driven scientific paradigm [20], this seems to be a significant gap and worthy of future attention.

Architectural discussions in this literature tend to focus either on the architecture of the PSE itself or the underlying infrastructure. In the former case, much attention is paid to component-based and compositional architectures, as illuminated by the focus on software components. In the case of the latter, as noted, there is a relationship between PSE research and grid computing that is evident, but in general, the emphasis on user interaction in the problem domain tends to intentionally abstract the underlying infrastructure away with references to grid computing, high-performance computing, or netcentric computing.

The objective of automating the creation or generation of problem-specific PSEs appeared later in the research. Among these papers, Cunha [10] discusses "building PSEs" in a general sense, but it wasn't until later that there were deeper investigations of this concept. For example, one approach, referred to a "meta-PSEs" [28], sought to create a PSE for solving the problem of creating a PSE. This is a powerful concept, because the goal is ultimately to deliver an end-to-end, problem-solving product to end users in their domains that minimizes the need for them to spend time doing anything other than working on their problem. A tool that can accept a specification for an environment that solves a class of problems and then deliver that environment would become a factory for maximizing problem-solving innovation.

1.3 History and Evolution of PSE Research

The spread of scientific computing in the 1980s, coupled with the increasing availability of desktop workstations with graphical user interfaces, gave rise to the idea of software environments that enable a user [3] to easily assemble powerful computational software around a problem specification, execute complex experiments consisting of potentially multiple simulation runs of a multidisciplinary collection of models, and then analyze, visualize and incorporate the results into the scientific body of knowledge. As mentioned above, the concept had sufficient traction by the first half of the 1990s to inspire several NSF workshops and a growing community of researchers, and Rice and Boisvert [44] comment that, by the middle of the 1990s, PSEs had "become 'expected' for large, sophisticated scientific software projects."

Houstis and Rice [21] track the early growth in their 2000 paper through a review of the publication dates of relevant papers in the bibliography of Gallopoulos, et. al. [22] (Figure 2). They note that the literature on PSEs generally falls into two categories: (1) application-oriented research detailing the development of a particular PSE in a problem-solving domain, and (2) more general discussions of PSE technology and infrastructure; the first category accounts for the larger portion of the literature and offers design and experiential studies of PSEs in various domains. A superficial review of the publications in this category reveals applications not only in the sciences, but also in engineering (e.g. chemical and aeronautical), decision support (e.g. construction, medicine, traffic management), mathematics (e.g. statistics and differential equation solvers, which are also common to many scientific applications), and analysis (e.g. geospatial, flood prediction, land use, and earthquakes). The second category is represented by the types of papers discussed previously and summarized in Table 1. While researchers publishing in the first category appear to come from a range of domains – often related to the classes of problems for which they have developed a PSE – researchers in the second category (as represented by the authors of the papers in Table 1) were generally involved in scientific, numerical, or mathematical computing; parallel, grid or high-performance computing; or data analysis, e-learning and other non-computer science applications of computing. Collaborations among these groups were common.

[3] These developments also spawned the field of end-user programming (EUP, also end-user development, EUD), which has a rich but curiously non-overlapping research literature with that of PSEs (only 0.5% of the papers in our Google Scholar search for this paper contained both terms). However, these are certainly related in their focus on the end user, their goals, and separation of domain and programming expertise. In the end-user development paradigm, PSEs generally follow the model-based development approach, though reuse and experiment design also align to parameterization and customization activities [32]. Interestingly, research into computation notebooks *has* been actively undertaken by the EUD community [31].

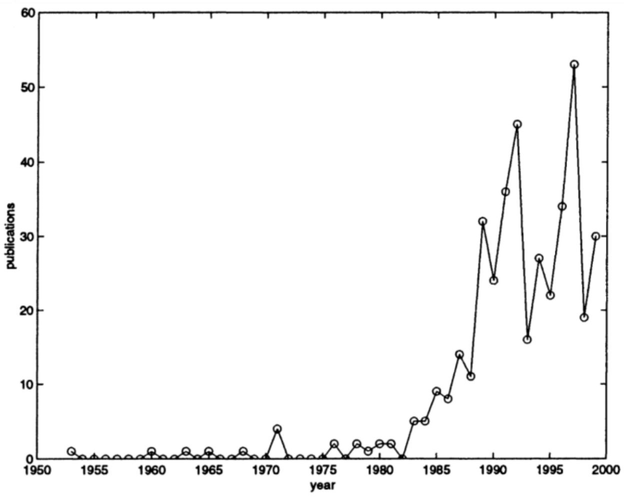

Figure 2. Early growth of PSE research based on bibliography in Gallopoulos [22], as presented in Houstis & Rice [21].

To build on this earlier study of the literature and understand what happened in the two decades since, publications data from Google Scholar were collected, resulting in the graph presented in Figure 3. Google Scholar (https://scholar.google.com) was queried for '("problem solving environments" | "problem solving environment")'. Additionally, because a large number of articles with these terms related to educational problem-solving environment research – a largely unrelated topic dealing with teaching problem-solving skills – the following exclusion terms were added: '-tutor -student -teach -education', which removed many of these. Year constraints were used to get specific values for each year between 1985 and 2022. The results still contain some potentially superfluous items, including foreign language documents that cannot be fully assessed; sampling suggests as much as 10% of the absolute results may not be relevant and therefore these numbers should not be used directly. Rather, the intent is to assess the overall trajectory of research publications relating to PSEs.

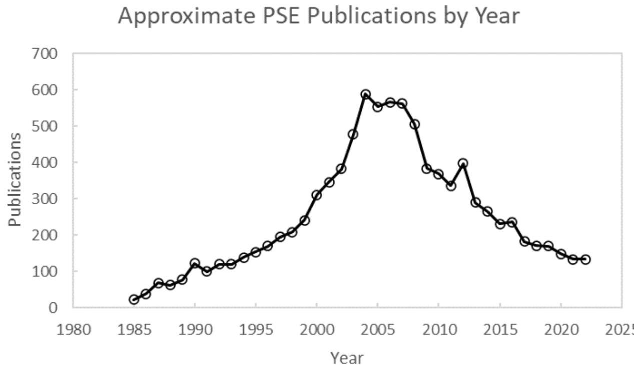

Figure 3. PSE publications based on Google Scholar analysis.

This expanded timeline reflects the continued growth in PSE research publications until the mid-2000s, but then a decline that continues until the present. Additionally, while multiple papers about PSE research (e.g. summaries, research agendas) were published in the years before the peak, none seem to appear in the recent literature. This begs the question: what happened? Did we achieve the long-term vision for PSEs by the 2010s?

2 Jupyter Rising

In 2007, Perez and Granger [37] formally introduced iPython, a "system for interactive scientific computing." Key features included an open programming environment that cleanly exposes all aspects of creating and executing scientific code, integration with GUIs and visualization libraries, and access to parallel and distributed backend computing infrastructure. At that time, the toolkit offered a typical, though fully interactive, code editing experience consisting (optionally) of an interactive terminal and graphical outputs that could be generated in separate window panes. Inspired by the highly integrated "notebook" experience found in products like Wolfram Mathematica[4], a web-based interactive notebook, initially iPython Notebook and later Jupyter Notebook [16], was added as the primary interface to the Jupyter "ecosystem" of scientific computing software. The transition to Jupyter also made the system programming language agnostic, inviting the statistical community with the open-source R language and newcomers such as Julia, among many others[5]. Since the mid-2010s, the ecosystem has grown not only in the scientific community, but beyond with the adoption of data scientific methods in all sectors; arguably, Jupyter is the interactive computing environment of choice today.

But, is Jupyter a PSE? Or a meta-PSE that allows construction of PSEs? Indeed, it has been argued that Jupyter is the path to realizing the vision for PSEs pursued by the PSE research community [4]. In this section, we will examine this critically, enumerating the features of computational notebooks, as well as some challenges with them that have emerged as they've been more broadly adopted. We then compare these to the key attributes of PSEs to assess the progress made along this path, toward informing and inspiring our concluding discussion of what additional research and development is required to truly realize the vision for PSEs.

2.1 Overview of Computational Notebooks

Computational notebooks provide the user with an interface that smoothly interleaves text, code and visualization in a linear flow of cells. This allows the user to develop the solution to a problem in pieces; each cell can contain a small piece of the code, and any output that it generates is presented immediately below it. Cells can also

[4] Wolfram Mathematica weaves an interesting thread through this topic. It – and tools like it such as Maple and Matlab – are found in the PSE literature as examples of general-purpose environments that can serve as PSEs. To the extent that mathematics serves as a *lingua franca* of scientific and engineering problem solving, they provide a tools supporting formulation and execution of problem solutions in this medium. Indeed, the description of Mathematica's Wolfram Language as a "computational language" goes beyond mathematics and brings us closer to direct representations of objects in the problem domain [51]. We also see these used in conjunction with other PSEs, e.g. SciRun [24]. Mathematica additionally provides one of the first notebook-style interfaces, foreshadowing the broad adoption of this design in computational notebooks.

[5] A current list of language kernels for Jupyter can be found here: https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

contain text entered by the user, allowing for *literate programming*, a concept developed by Donald Knuth to, in part, provide more naturally described and organized programs [27]. Importantly, though, computational notebooks don't enforce a literate programming – or generally any – paradigm, but rather are relatively freeform scratchpads for computational work; this general-purpose flexibility arguably makes them desirable to problem solvers, in much the same way that spreadsheets are.

While Jupyter is the most commonly encountered computational notebook, others have emerged with variations on the key features of the genre. Lau, et. al. [31] enumerate the design space created by considering 60 notebooks, identifying the following common feature categories:

- Access to data sources
- Code editing in one or more programming languages (includes coding-focused collaboration)
- Code execution and results visualization
- Publishing from the notebook, enabled by the explication of code and inclusion of explanatory prose

Three key objectives for computational notebooks are accessibility, sharing and reproducibility [26]. The notebook interface was introduced to iPython in part to make it easier to interact with computation. As a web-based application, it is easily hosted and accessed through a web browser, and this same technology allows local desktop installation.

The notebooks enable sharing in several ways: the underlying notebook itself can be shared and opened in Jupyter or a compatible environment, or the notebook can be exported as HTML and opened in any browser (even as a local file). Moreover, this kind of sharing can retain interactivity for some elements of the notebook, allowing recipients to explore the results (computations generally cannot be re-executed as the computational kernel doesn't accompany the notebook). The notebook file also contains the code, enabling code reuse, though it should be noted that this differs from typical software reuse in that it would be reuse of the notebook (i.e. as a starting point for changing or adding computations) or code snippets within it rather than as a software library. The idea of using notebooks for sharing methods and results can be extended to domains beyond science (e.g. systems engineering [56]).

The ability to share notebooks in these ways facilitates scientific reproducibility. The code and the results can be easily examined, and more importantly, the interweaved prose enables both to be explained, forming a *computational narrative* [25]. At a minimum, this supports traditional publishing in the sciences, but because code is available, and provided the right data and environment are also available, the computations can be re-executed.

Together, these attributes make a compelling environment for scientists and others to use for doing and sharing their computational work, but how does that work in practice?

2.2 Emerging Challenges with Notebooks

As notebooks have become more widely used, both within scientific computing and beyond, limitations in their design and use have emerged [17]. In some cases, these are a result of design tradeoffs made for simplicity and flexibility, which have fueled broader adoption and unexpected applications. In others, how they are used in practice does not match the vision in their design [41]. Chattopadhyay, et. al. [8] identify several issues in their study of notebook users' pain points that are relevant to our discussion of PSEs, including:

- Loading data, especially across sources and platforms, is very difficult and time consuming
- Constantly "tweaking" code and latency in the feedback from the changes is frustrating
- Non-linear execution order hampers debugging, version control, and user understanding of program state
- Code management and the complexity arising from library dependencies requires software engineering skills
- Sharing relevant parts of notebooks is difficult and limited
- Reproducibility is hampered by individual customizations and complicated dependencies
- Deploying a notebook as a product of scientific work, or a tool to be used by others, is beyond most users' skills

If a significant objective of literate programming and computational notebooks is explanation via computational narratives, in practice, few scientists seem to do this. Rule, Tabard and Hollan [46] looked at over a million notebooks available online, finding that 25% contained no narrative at all. In interviewing notebook users, they found most use them for "personal, exploratory, and messy" computations – in other words, without the intent to communicate and share with others.

3 Have We Achieved the Goals of a PSE?

Given the features and flexibility of computational notebooks, but with consideration of their limitations in practice, how well do they satisfy the goals of PSEs? In this section, we discuss each of the PSE key features derived in section 1.1 in relation to how they are approached – and whether they are achieved – by the current generation of computational notebooks.

3.1 Problem-domain interaction

The objective in problem-domain interaction is to enable a user to work in the language of their domain and in a manner analogically consistent with the domain. For example, a chemist would most likely want to work with chemical formulae, and models of chemical processes, which could be integrated in a natural way. Computational notebooks, though, tend to focus primarily on traditional, text-based programming languages with occasional support for domain-relevant notation. Support could be provided in part by domain-specific languages or language kernels, but domain-oriented interaction patterns would be more difficult to layer over the notebook-style design. Notably, alternatives to the notebook structure have emerged to explore other potential representations [50].

3.2 Multidiscipline and collaboration support

In this category, we capture both the need for multiple disciplines to work together on complex problems (*horizontal collaboration*), as well as collaboration in the broader sense, to include among domain experts, data scientists, and software engineers (*vertical collaboration*). Again, to the degree that domain-centric interactivity is not well supported, multidisciplinary integration must occur through a *lingua franca* in the notebook environment. This often takes the form of the dominant programming language. Higher-level integrations among libraries and services are possible, but also only through active translation to the underlying programming environment.

Collaboration vertically with other technical disciplines is generally accomplished through code reuse; a computer scientist may develop a library implementing an optimized version of some computational function that the scientist can then make use of. Some environments have experimented with built-in, explicit collaboration support [7], though this is atypical.

3.3 Intelligent support throughout the problem-solving process

Current notebooks focus on intelligent support to the programming process, borrowing features from mainstream integrated development environments like code completion. However, there is almost no support for problem domain-centric intelligence, again due to their relatively domain-agnostic design. This is an area of active research with work looking at various parts of the problem-solving process to include data wrangling [38], coding [34], and presentation [57].

3.4 Problem-solving knowledge capture and sharing

The literate programming-inspired design of the notebook interface provides a mechanism for problem-solving knowledge capture, though as noted earlier, many users do not effectively make use of this to annotate their notebooks. Moreover, because the notebook's structure enforces a linear presentation of the realized problem-solving process, without explicit intent on the part of the user, capturing this process (vice just the resulting product) is additional work, for which the notebook offers little support. Some research toward automatically capturing provenance from scripts and script execution has been done [40].

3.5 Software sharing and reuse

Notebooks provide a relatively accessible interface to multiple ecosystems of reusable software libraries. As computational solutions to problems are rendered as software in the notebook, the ability to share and reuse is well supported; as noted above, multiple modes of sharing are offered, though underlying language ecosystem challenges with dependency management, as well as an inability to easily share a portion of a notebook – aside from copy-and-paste of raw code – are shortcomings.

3.6 Components, component types and integration

The Jupyter ecosystem, which was designed for extensibility, offers a large range of existing components, primarily in the form of computational components, visualizations, and interactive (user interface) components. Computational components generally derive from software libraries imported to the programming environment, though the ability to copy and paste cell-based code also represents a weak type of component-based sharing. However, incorporating components can vary from a straight-forward import to a tremendous challenge; component use is not universally the plug-and-play experience that is desired.

Integrating components – enabling them to work together – can also be less than seamless. Part of this stems from lack of clarity in the underlying computation mechanism and how data is shared among the elements of a notebook between the kernel backend and the notebook frontend. While usually possible, more complex integrations require additional technical knowledge. Customizations made to support various components and their integration may also be less portable, affecting sharing and reproducibility.

3.7 Experiment management

To the extent that notebooks support component integration, they also support do-it-yourself computational experiment design and execution. Users generally have to write their own code to implement an experimental setup, especially if there is need to iterate over a latin square of experimental parameters; notebooks contain no intrinsic mechanisms for organizing or capturing experiment results beyond what is returned in a cell.

3.8 Computing architecture and infrastructure

An ecosystem like Jupyter offers both a system-level architecture – web-based notebooks hosted on a collaborative server with support for one or more language kernels – and access to computing infrastructure. In some cases, this is accomplished through the libraries made available in the programming environment (at which point the user is on their own to leverage the infrastructure through the library interface), and some infrastructure developers and cloud compute providers have integrated notebooks into their offerings.

3.9 Summary

Overall, we observe that computational notebooks excel at providing easy access to programming languages, software libraries, visualization components, and the ability to annotate, share, reuse and present computational research. However, this is largely accomplished in the domains of computer and data science; notebook users must be literate in a programming language and the peculiarities of modern software development to fully leverage these features. As far as problem definition, notebooks do provide their users a blank, linear slate on which they can develop and document their problem's structure and problem-solving process, but will offer them little support or incentive for doing this.

So, it is arguable that, under the generalization of "providing all computational facilities necessary to solve a class of problems," notebooks do achieve this in the computational sense (that is, like our hypothetical "laptop with a development tool"), and their substantial adoption may testify to this as a practical matter. However, as we look deeper at the intent motivating PSEs, we find a significant gap between domain-centric problem solving and the facilities current computational notebooks offer.

4 A Refocused Future for PSE Research

Given the remaining gaps between the vision for PSEs and the achievements of notebooks as the most widely adopted user-facing tools for computational science, where should we focus our future research and development efforts, and what is to be gained in doing so? We propose to focus on two features discussed earlier that offer the greatest opportunity: interacting in the problem-domain language and intelligent support throughout the problem-solving process, and seek to find a solution in their integration with current notebook technologies.

4.1 Interaction in the Problem-Domain Language

The first major area that remains largely unaddressed is in enabling problem solvers to work in the native languages of their domains. This is certainly challenging in that each domain has its own language, concepts and structures. An approach to this is leveraging conceptual models, which have been touched on in the context of PSEs [49], but have gained significant traction in the closely allied field of modeling and simulation [36]. Robinson, et. al. [45] discuss this application, noting that "conceptual modeling is about moving *from a problem situation*, through model requirements to a definition of *what* is going to be modeled and *how*" (p. 10, emphasis mine) and "Conceptual modeling is about determining the right model, not how the software will be implemented" (p. 13). This framing – focusing on modeling the problem in the language of the domain and in a manner agnostic to the eventual software implementation – is the same as we seek in a PSE (unsurprisingly, as many PSEs reported in the literature *are* modeling and simulation environments).

Conceptual modeling languages, which are often graphical, provide a common means for expressing objects and actions in any domain and making them accessible and potentially reusable across domains, which would facilitate both horizontal (multidisciplinary) and vertical collaboration. Indeed, conceptual modeling approaches such as Object-Process Modeling [11], designed for systems engineering, have demonstrated successful application in other domains, for example, molecular biology [12]. Integrating conceptual modeling methods and tools with notebook technologies would provide an environment in which conceptual models expressing a problem and desired outcome in the problem domain could be directly connected to software implementations via notebook code. This would inherently associate richer descriptions of hypotheses, models, data, and the relationships among them from the conceptual models with the underlying code, which will in turn support better understanding and communication of the scientific and problem-solving knowledge artifacts resulting from the computational study.

4.2 Intelligent Support for Problem Solving

The other significant aspect of the vision that remains unrealized is intelligent support throughout the problem-solving process. Earlier PSE work in this area largely focused on recommendation engines that would help the user locate reusable software assets to compose into a solution. The growth in web-based software repositories makes software sharing and access easier, but a problem that remains is the ability to find potentially useful software *across* domains. Notably, current notebooks do not contain integrated recommender technologies; their addition would begin to simplify the process of finding and integrating software. Progress on cross-domain software search has been made through the application of machine learning, e.g. [58], and if we were to also provide richer domain context through associated conceptual models (which, themselves can help facilitate cross-domain sharing), this would be further improved.

Substantial advances in artificial intelligence (AI) in recent years offer a variety of opportunities far exceeding the original vision for intelligent support in PSEs. For example, the abilities of large language models to write narrative, write code, document code, perform data analysis, and use and coordinate external tools [6] presents compelling possibilities for adding value to notebooks. This class of AI alone has the potential to touch most of the key phases in the problem-solving life cycle, and early experiments with such integrations have proved promising [55]. Combined with the richer descriptions of scientists' and engineers' intentions made available to AI technologies through the explicit representation of the problem/solution interface with conceptual models will make such integrations more effective.

5 Conclusion

The broad adoption of computational notebooks has made progress toward the goals for PSEs, but there is more to do to achieve the vision. It remains a worthy vision: better structuring the problem-solving process by integrating conceptual models with computational notebooks, improving collaboration and communication within and across disciplines, and leveraging rapidly emerging artificial intelligence technologies all offer mechanisms for reducing problem-solving friction so that practitioners can focus on the core of their work. These technologies are, by themselves, either relatively mature or receiving a great deal of research attention. The time is right to build on these successes by driving research and development toward the integration of these solutions to address the missing pieces of the PSE vision and seamlessly coupling them with proven computational notebook technologies to create truly general-purpose problem-solving environments.

Acknowledgments

The author would like to thank his colleagues and the reviewers for their suggestions, all of which greatly improved this paper.

References

[1] J.E. Allen, C.I. Guinn, and E. Horvitz. 1999. Mixed-initiative interaction. *IEEE Intelligent Systems* 14, 5: 14–23. https://doi.org/10.1109/5254.796083

[2] B. Appelbe, L. Moresi, S. Quenette, and P. Simter, "Scientific Software Frameworks and Grid Computing," in *Grid-Based Problem Solving Environments*, P. W. Gaffney and J. C. T. Pool, Eds., in IFIP The International Federation for Information Processing, vol. 239. Boston, MA: Springer US, 2007, pp. 401–413. doi: 10.1007/978-0-387-73659-4_24.

[3] Bajaj, Chanderjit; Hoffmann, Christoph M.; Houstis, Elias N.; Korb, John T.; and Rice, John R., "Computing About Physical Objects" (1987). *Department of Computer Science Technical Reports.* Paper 603.

https://docs.lib.purdue.edu/cstech/603

[4] Lorena A. Barba. 2021. The Python/Jupyter Ecosystem: Today's Problem-Solving Environment for Computational Science. *Computing in Science & Engineering* 23, 3: 5–9. https://doi.org/10.1109/MCSE.2021.3074693

[5] J. L. Barros-Justo, F. B. V. Benitti, and S. Matalonga, "Trends in software reuse research: A tertiary study," *Computer Standards & Interfaces*, vol. 66, p. 103352, Oct. 2019, doi: 10.1016/j.csi.2019.04.011.

[6] Daniil A. Boiko, Robert MacKnight, and Gabe Gomes. 2023. Emergent autonomous scientific research capabilities of large language models. Retrieved April 15, 2023 from http://arxiv.org/abs/2304.05332.

[7] Niels Olof Bouvin. 2019. From NoteCards to Notebooks: There and Back Again. In *Proceedings Conference on Hypertext and Social Media*, 19–28. https://doi.org/10.1145/3342220.3343666

[8] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. 2020. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (CHI '20), 1–12. https://doi.org/10.1145/3313831.3376729

[9] M. F. Costabile, D. Fogli, C. Letondal, P. Mussio, and A. Piccinno, "Domain-Expert Users and their Needs of Software Development," p. 5.

[10] José C. Cunha. 2001. Future Generations of Problem—Solving Environments. In *The Architecture of Scientific Software*, Ronald F. Boisvert and Ping Tak Peter Tang (eds.). Springer US, Boston, MA, 29–37. https://doi.org/10.1007/978-0-387-35407-1_2

[11] Dov Dori. 2016. *Model-Based Systems Engineering with OPM and SysML*. Springer New York, New York, NY. https://doi.org/10.1007/978-1-4939-3295-5

[12] Dov Dori and Mordechai Choder. 2007. Conceptual Modeling in Systems Biology Fosters Empirical Findings: The mRNA Lifecycle. *PLoS ONE* 2, 9: e872. https://doi.org/10.1371/journal.pone.0000872

[13] I. Foster, M. E. Papka, and R. Stevens, "Tools for distributed collaborative environments: a research agenda," in *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing HPDC-96*, Syracuse, NY, USA: IEEE, 1996, pp. 23–28. doi: 10.1109/HPDC.1996.546170.

[14] P. W. Gaffney, J. C. T. Pool, and IFIP Working Group 2.5--Mathematical Software, Eds., Grid-based problem solving environments: IFIP TC2/WG 2.5 Working Conference on Grid-based Problem Solving Environments: implications for development and deployment of numerical software, July 17-21, 2006, Prescott, Arizona, USA. in IFIP, no. 239. New York: Springer, 2007.

[15] D. Gannon, M. Christie, S. Marru, S. Shirasuna, and A. Slominski, "Programming Paradigms for Scientific Problem Solving Environments," in *Grid-Based Problem Solving Environments*, P. W. Gaffney and J. C. T. Pool, Eds., in IFIP The International Federation for Information Processing, vol. 239. Boston, MA: Springer US, 2007, pp. 3–15. doi: 10.1007/978-0-387-73659-4_1.

[16] Brian E. Granger and Fernando Perez. 2021. Jupyter: Thinking and Storytelling With Code and Data. *Computing in Science & Engineering* 23, 2: 7–14. https://doi.org/10.1109/MCSE.2021.3059263

[17] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. 2019. Managing Messes in Computational Notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*, 1–12. https://doi.org/10.1145/3290605.3300500

[18] S. Henn, *When Women Stopped Coding*. in NPR:Planet Money. NPR, 2014. Accessed: Jun. 25, 2023. [Online]. Available: https://www.npr.org/sections/money/2014/10/21/357629765/when-women-stopped-coding

[19] Thomas T. Hewett and Jennifer L. DePaul. 2000. Toward a Human Centered Scientific Problem Solving Environment. In *Enabling Technologies for Computational Science*, Elias N. Houstis, John R. Rice, Efstratios Gallopoulos and Randall Bramley (eds.). Springer US, Boston, MA, 79–90. https://doi.org/10.1007/978-1-4615-4541-5_7

[20] Tony Hey, Stewart Tansley, and Kristen Tolle (eds.). 2009. *The fourth paradigm: data-intensive scientific discovery*. Microsoft Research, Redmond , Washington.

[21] Elias N. Houstis and John R. Rice. 2000. Future problem solving environments for computational science. *Mathematics and Computers in Simulation* 54, 4–5: 243–257. https://doi.org/10.1016/S0378-4754(00)00187-7

[22] E. Houstis, E. Gallopoulos, R. Bramley, and J. Rice. 1997. Problem-solving Environments For Computational Science. IEEE Computational Science and Engineering 4, 3: 18–21. https://doi.org/10.1109/MCSE.1997.615427

[23] Elias N. Houstis, John R. Rice, Efstratios Gallopoulos, and Randall Bramley (eds.). 2000. *Enabling Technologies for Computational Science*. Springer US, Boston, MA. https://doi.org/10.1007/978-1-4615-4541-5

[24] C. Johnson, S. Parker, D. Weinstein, and S. Heffernan, "Component-based, problem-solving environments for large-scale scientific computing," *Concurrency Computat.: Pract. Exper.*, vol. 14, no. 13–15, pp. 1337–1349, Nov. 2002, doi: 10.1002/cpe.693.

[25] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. 2018. The Story in the Notebook: Exploratory Data Science using a Literate Programming Tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*, 1–11. https://doi.org/10.1145/3173574.3173748

[26] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Matthias Bussonnier, Jonathan Frederic, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Safia Abdalla, and Carol Willing. Jupyter Notebooks—a publishing format for reproducible computational workflows.

[27] Donald Ervin Knuth. 1992. *Literate programming*. Center for the Study of Language and Information, Stanford, Calif.

[28] H Kobashi, S Kawata, Y Manage, M Matsumoto, H Usami, and D Barada. 2010. A meta Problem Solving Environment (PSE). In *5th International Conference on Computer Sciences and Convergence Information Technology*, 253–259. https://doi.org/10.1109/ICCIT.2010.5711067

[29] Z. Kulpa, M. Sobolewski, and S. N. Dwivedi, "Graphical User Interface with Object-Oriented Knowledge-Based Engineering Environment," in *CAD/CAM Robotics and Factories of the Future '90*, S. N. Dwivedi, A. K. Verma, and J. E. Sneckenberger, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 154–159.

[30] D. Lancaster and J. S. Reeve, "Computational Steering in Problem Solving Environments," in *Euro-Par 2000 Parallel Processing*, A. Bode, T. Ludwig, W. Karl, and R. Wismüller, Eds., in Lecture Notes in Computer Science, vol. 1900. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1340–1344. doi: 10.1007/3-540-44520-X_187.

[31] Sam Lau, Ian Drosos, Julia M. Markel, and Philip J. Guo. 2020. The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 1–11. https://doi.org/10.1109/VL/HCC50065.2020.9127201

[32] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-User Development: An Emerging Paradigm," in *End User Development*, H. Lieberman, F. Paternò, and V. Wulf, Eds., Dordrecht: Springer Netherlands, 2006, pp. 1–8. doi: 10.1007/1-4020-5386-X_1.

[33] B. Ludäscher *et al.*, "Scientific workflow management and the Kepler system," *Concurrency Computat.: Pract. Exper.*, vol. 18, no. 10, pp. 1039–1065, Aug. 2006, doi: 10.1002/cpe.994.

[34] A. M. Mcnutt, C. Wang, R. A. Deline, and S. M. Drucker, "On the Design of AI-powered Code Assistants for Notebooks," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, Hamburg Germany: ACM, Apr. 2023, pp. 1–16. doi: 10.1145/3544548.3580940.

[35] M. J. O'Connor, C. Nyulas, S. Tu, D. L. Buckeridge, A. Okhmatovskaia, and M. A. Musen, "Software-engineering challenges of building and deploying reusable problem solvers," *AIEDAM*, vol. 23, no. 4, pp. 339–356, Nov. 2009, doi: 10.1017/S0890060409990047.

[36] Dale K Pace. 2000. Ideas About Simulation Conceptual Model Development. *JOHNS HOPKINS APL TECHNICAL DIGEST* 21, 3.

[37] Fernando Perez and Brian E. Granger. 2007. IPython: A System for Interactive Scientific Computing. *Computing in Science & Engineering* 9, 3: 21–29. https://doi.org/10.1109/MCSE.2007.53

[38] T. Petricek, G. J. J. Van Den Burg, A. Nazabal, T. Ceritli, E. Jimenez-Ruiz, and C. K. I. Williams, "AI Assistants: A Framework for Semi-Automated Data Wrangling," *IEEE Trans. Knowl. Data Eng.*, pp. 1–12, 2022, doi: 10.1109/TKDE.2022.3222538.

[39] S. Picard, J.-L. Ermine, and B. Scheurer, "Knowledge Management for Large Scientific Software".

[40] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "noWorkflow: a tool for collecting, analyzing, and managing provenance from python scripts," *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 1841–1844, Aug. 2017, doi: 10.14778/3137765.3137789.

[41] Joao Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 507–517. https://doi.org/10.1109/MSR.2019.00077

[42] J. C. Pitt. 2001. “What Engineers Know,” *Techné: Research in Philosophy and Technology*, 5, 3: 116–123. doi: 10.5840/techne2001532.

[43] J.R. Rice. 1999. A perspective on computational science in the 21st Century. *Computing in Science & Engineering* 1, 2: 14–16. https://doi.org/10.1109/5992.753042

[44] J.R. Rice and R.F. Boisvert. 1996. From scientific software libraries to problem-solving environments. *IEEE Computational Science and Engineering* 3, 3: 44–53. https://doi.org/10.1109/99.537091

[45] Stewart Robinson, Roger Brooks, Kathy Kotiadis, and Durk-Jouke Van Der Zee. 2010. *Conceptual modeling for discrete-event simulation.* CRC Press.

[46] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*, 1–12. https://doi.org/10.1145/3173574.3173606

[47] C. A. Shaffer, L. T. Watson, D. G. Kafura, and N. Ramakrishnan. 2000. “Features of Problem Solving Environments for Computational Science,” in *Proceedings of the 2000 High Performance Computing Symposium (HPC'00)*, San Diego, CA: Society for Computer Simulation International: 242–247.

[48] P. E. van der Vet *et al.*, “Smart Environments for Collaborative Design, Implementation, and Interpretation of Scientific Experiments,” *Workshop on AI for Human Computing (AI4HC)*, pp. 79–86, Jan. 2007.

[49] Marc Vass, John M. Carroll, and Clifford A. Shaffer. 2002. Supporting creativity in problem solving environments. In *Proceedings of the fourth conference on Creativity & Cognition - C&C '02*, 31–37. https://doi.org/10.1145/581710.581717

[50] Zijie J. Wang, Katie Dai, and W. Keith Edwards. 2022. StickyLand: Breaking the Linear Presentation of Computational Notebooks. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 1–7. https://doi.org/10.1145/3491101.3519653

[51] S. Wolfram, “What We've Built is a Computational Language (and That's Very Important!),” *Stephen Wolfram Writings*, May 09, 2019. https://writings.stephenwolfram.com/2019/05/what-weve-built-is-a-computational-language-and-thats-very-important/ (accessed Jul. 09, 2023).

[52] S. Wolfram, *Celebrating 35 Years of Mathematica*, (Jun. 23, 2023). Accessed: Jul. 02, 2023. https://www.youtube.com/watch?v=HxWg8exJxNY&t=1240s

[53] K. Wolstencroft *et al.*, “The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud,” *Nucleic Acids Research*, vol. 41, no. W1, pp. W557–W561, Jul. 2013, doi: 10.1093/nar/gkt328.

[54] M. Yarrow, K. M. McCann, R. Biswas, and R. F. Van Der Wijngaart, “An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid,” in *Grid Computing — GRID 2000*, R. Buyya and M. Baker, Eds., in Lecture Notes in Computer Science, vol. 1971. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 146–157. doi: 10.1007/3-540-44444-0_14.

[55] Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, Alex Polozov, and Charles Sutton. 2022. Natural Language to Code Generation in Interactive Data Science Notebooks. Retrieved April 12, 2023 from http://arxiv.org/abs/2212.09248

[56] Jack Zentner and Tom McDermott. 2017. Web notebooks as a knowledge management tool for system engineering trade studies. In *2017 Annual IEEE International Systems Conference (SysCon)*, 1–5. https://doi.org/10.1109/SYSCON.2017.7934710

[57] C. Zheng, D. Wang, A. Y. Wang, and X. Ma, “Telling Stories from Computational Notebooks: AI-Assisted Presentation Slides Creation for Presenting Data Science Work,” in *CHI Conference on Human Factors in Computing Systems*, New Orleans LA USA: ACM, Apr. 2022, pp. 1–20. doi: 10.1145/3491102.3517615.

[58] Feng Zhu, Yan Wang, Chaochao Chen, Guanfeng Liu, Mehmet Orgun, and Jia Wu. 2018. A Deep Framework for Cross-Domain and Cross-System Recommendations. *In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 3711–3717. https://doi.org/10.24963/ijcai.2018/516